GEORGIA INSTITUTE OF TECHNOLOGY — OFFICE OF CONTRACT ADMINISTRATION

PROJECT ADMINISTRATION DATA SHEET

[x] ORIGINAL [ ] REVISION NO. ____

Project No. G-36-610 GTRI/~~GTX~~ DATE 6/8/83

Project Director: A. P. Jensen School/~~Lab~~ ICS

Sponsor: MERADCOM, Procurement & Production Directorate, Ft. Belvoir, VA

Type Agreement: D. O. #0016 under BOA DAAK70-79-D-0087 (AIRMICS) (OCA File #42)

Award Period: From 6/1/83 To ~~5/31/84~~ 6-16-84 (Performance) --- (Reports)

Sponsor Amount:	This Change	Total to Date
Estimated:	$ 145,000	$ 145,000
Funded:	$ 145,000	$ 145,000

Cost Sharing Amount: $ None Cost Sharing No: N/A

Title: Distributed Combat Service Support Advanced Experimental Demonstrations

ADMINISTRATIVE DATA OCA Contact William F. Brown Ext. 4820

1) Sponsor Technical Contact:

Major David Forinash
AIRMICS
115 O'Keefe Bldg.
Atlanta, GA 30332

(404) 894-3101

2) Sponsor Admin/Contractual Matters:

T. A. Bryant
ONR RR
Campus
(404) 881-4213

see Rev # 2

Defense Priority Rating: DO-S1 Military Security Classification: None

(or) Company/Industrial Proprietary: --

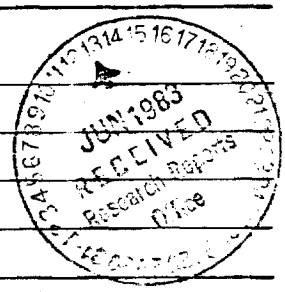
RESTRICTIONS

See Attached Gov't Supplemental Information Sheet for Additional Requirements.

Travel: Foreign travel must have prior approval – Contact OCA in each case. Domestic travel requires sponsor approval where total will exceed greater of $500 or 125% of approved proposal budget category.

Equipment: Title vests with Government

COMMENTS:



COPIES TO:

Project Director
Research Administrative Network
Research Property Management
Accounting

Procurement/EES Supply Services
Research Security Services
Reports Coordinator (OCA)
Research Communications (2)

GTRI
Library
Project File
Other I. Newton

RM-OCA-4-283

GEORGIA INSTITUTE OF TECHNOLOGY OFFICE OF CONTRACT ADMINISTRATION

SPONSORED PROJECT TERMINATION/CLOSEOUT SHEET

3279

Date October 2, 1985

roject No. G-36-610 School ICS

cludes Subproject No.(s) N/A

oject Director(s) A.P. Jensen GTRC /

onsor U.S. Army Computer Systems Command, Ft. Belvoir, VA.

le Distributed Combat Service Support Advanced Experimental Demonstrations

ective Completion Date: 7/10/84 (Performance) (Reports)

nt/Contract Closeout Actions Remaining:

- [ ] None
- [X] Final Invoice or Final Fiscal Report
- [ ] Closing Documents
- [X] Final Report of Inventions
- [X] Govt. Property Inventory & Related Certificate
- [ ] Classified Material Certificate
- [ ] Other

inues Project No. Continued by Project No.

ES TO:

ct Director
rch Administrative Network
rch Property Management
unting
rement/GTRI Supply Services
rch Security Services
Coordinator, IOCA
Services

Library
GTRC
Research Communications (2)
Project File
Other Heyser

Jones

OCA 69.285

Combat Service Support System
Advanced Experimental Demonstration

Presented to:

The
Army Institute for Research
in
Management Information and Computer Systems
(AIRMICS)

Contract No. DAAK70-79-D-0087-0016
Research Project No. G36-610

Principal Investigator:

Alton P. Jensen, Professor

Project Manager:

William O. Putnam, Research Scientist II

Principal Project Staff:

Steven L. Goldberg, Research Assistant
I-Ming Pai, Graduate Research Assistant
Miri So, Graduate Research Assistant

School of Information and Computer Science

GEORGIA INSTITUTE OF TECHNOLOGY

Atlanta, Georgia 30332

July 17, 1984

DISCLAIMER

The findi s of this r ort are not to be construed as an official Departmen of the Ar position, l less so designated by other authorized documents.

Table of Contents

Part 1. Project Overview

In the year ending July 17, 1984, the project group made good progress in establishing a foothold on the problems of networking networks of heterogeneous computers via asynchronous dial-up lines. In these endeavors, fundamental issues such as network interfaces, automatic dialing, message routing and rerouting, message forwarding, and error detection have been addressed.

The year was concluded with successful demonstrations of a network incorporating the above mentioned features. The network consisted of an ONYX computer (running UNIX), a Honeywell DPS-6 (running GCOS), and an IBM Series/1 (running EDX).

The Honeywell DPS-6 is designated as the DAS3 in the AIRMICS network plan. The ONYX was used as the TACCS. The Series/1 was used to demonstrate the ability to manage communications between radically different systems.

At the beginning of the 1983-84 year, the specifications for the Advanced Experimental Demonstrations (AED) were predicated on using three IBM Series/1 computers and a Honeywell DPS-6. The Honeywell had been specified to represent the DAS3 in the network simulation. The Series/1 machines were to be used to represent the TACCS.

A Honeywell DPS-6 was ordered and installed in December of 1983. In the meantime, work proceeded on high level system design and specification of the network interface.

It was determined that the software support provided by the Series/1 EDX system would be inadequate for the desired system. It was necessary to write a device handler for TTY type terminals in order to support character-by-character, full duplex, buffered, asynchronous communications. This device handler permitted a cleaner and more generic system implementation. Once it was complete, it was possible to develop a system in which messages and files were passed among a group of three Series/1 computers, an IBM 4341, a CDC CYBER, and a PRIME 550.

Presentations on the high-level design and implementation decisions were given and the system was demonstrated for AIRMICS personnel in September, 1983.

In this demonstration, one facet of the AED was omitted. The links between machines were supposed to be dial-up lines in the standard telephone network. The auto-dial modems necessary for this part of the demonstration were not available in time, so the connections were made directly. Arrangements were made for modems to be available for the rest of the year pursuant to plans for the next demonstration.

Subsequently, the three Series/1 machines were interconnected via the "Chat Ring" IBM hardware and the IBM register insertion protocol implemented under the Communications Facility II programs. This network was interfaced to the

Ungermann Bass Net/One LAN through one Series/1 acting as a host system. In this environment, the previous demonstrations were repeated and augmented with complete IBM-3270 protocol emulation using ASCII terminals under the YALE ASCII programs via a Net/One connection. This demonstration was in itself a first in that one Series/1 machine was channel connected to the IBM-4341 host system while another Series/1 machine (using the YALE ASCII system) served as a terminal concentrator for ASCII terminals connected via Net/One simulating 3270 workstations on the 4341 host system.

By January, the Honeywell DPS-6 had arrived but had not passed acceptance tests. The system development on the Series/1 proceeded with activity concentrating on the dialing, connection, and routing algorithms. At the end of January, 1984, the Honeywell system was accepted. Training and orientation proceeded.

In February, it became clear that the Series/1 with EDX would not be used by the Army as a TACCS. The TACCS would be some type of 16 bit microcomputer running UNIX. Emphasis was shifted to the Honeywell and ONYX computers now available at the AIRMICS installation. Some of the Series/1 software was modified and retained to interconnect with the other two systems for demonstration purposes.

High level design for the network was completed in March, 1984. An overview of the design is presented in Part 3.

In order to conserve effort and to insure portability of code, software development was shifted to the ONYX system using the UNIX operating system and the "C" language. It had become clear that the TACCS would be a UNIX based micro similar to the ONYX. Since the Series/1 EDX code was to be abandoned at the completion of this demo, new development was suspended.

The system design was refined and the system implemented. The system was in place by late May and final testing for the close-out demonstration was completed.

This demonstration illustrated interactive or batch submission of messages and files to be transferred among three machines: one Series/1, one ONYX, and the Honeywell DPS-6. Automatic forwarding and routing were performed. Utilities were provided for the generation and display of messages. All linkages between machines were made automatically through the telephone network using 1200 bps auto-dial modems. Due to an operating system bug on the ONYX, the ONYX could transmit data to the Series/1 properly, but could not receive files from that machine. All other links worked as designed. An initial demonstration was given for local AIRMICS personnel during the first week of June.

Demonstrations for external reviewers were given during the next week and last week of June. The system performed according to specifications. Several presentations were made by group members on the design and development of the system.

Part 2. Considerations in software design

During the development of the software necessary for the AEDs of the current contract and the issues of simulating command and control environments, a number of challenges were identified. This part of the report will briefly review these areas in order to convey the nature of the tasks to be accomplished and to identify progress that has been made.

The Honeywell DPS6 (as a DAS3) uses the GCOS 400 operating system which has been described by some as "UNIX-like." However, GCOS has none of the standard UNIX features such as pipes, I/O redirection, user-readable directories, or dynamic file binding. The system does not provide the same power as UNIX except at an assembly language programming level. The high level languages such as PASCAL, provide none of the system level facilities of the language C.

The program development environment provided by GCOS has known deficiencies: the editor inserts spurious characters in program source files resulting in compiler errors and increased debugging time; the GCOS PASCAL compiler has a number of bugs which cause it to return errors in correct programs often making it necessary to recompile the same program two or three times (with no modifications) before the compiler produces correct object code; the linker also has trouble with the compiled PASCAL programs so it is often necessary to link a program several times to get an executable program module.

Another problem concerned the handling of ports by GCOS. A port must be in one of two states: LISTENUR-configured or NON-LISTENUR-configured. If the port is not configured for the LISTENUR process, users cannot log in on the port. However, if it is so configured, a program cannot use the port to dial out on a modem. This means that a single port cannot be used for both dial-out and dial-in operation. The Honeywell thus requires twice as many ports and modems as the other systems. UNIX handles ports in a similar manner, but the nature of the system is such that it is easy to write code (in the high level language C) to circumvent the problem.

This project is directed to building a communications network which allows the transfer of files and messages among the Honeywell, ONYX, and IBM Series/1 machines. These machines all have very different operating system and hardware characteristics but the machine dependent elements of software in this system are restricted to the small low level parts of the code while the high level interface among the various computers is machine independent. This structure provides for ease of expansion and maintenance.

At the hardware level, it was necessary to deal with three different types of modems as well as three distinct and different machine interfaces:

* Each type of modem requires special code to handle the auto-dial procedures therefore it is necessary to keep track of which type of modem is in use at each machine.

* There are also differences in parity requirements for each kind of computer: the Honeywell sends and receives with either odd or even parity only (mark, space, and ignore parity options are not allowed); the Series/1's transmit space parity, and ignore input parity; the ONYX systems can be configured for any type of parity, but only for one type at a time.

* Since the AED requires the other computers to call the ONYX, and since the ONYX has no way of knowing which one is calling, it cannot set the parity of the port in advance. If the parity is not set, the calling machine cannot communicate.

* The Cermetek modem used on the ONYX recognizes the parity of the ONYX port and throws away all incoming characters which are not of that parity, making it impossible for the caller to tell the ONYX which parity to use.

At the high level interface, there were also problems to be solved. It was necessary to allow an operator to type and send a message interactively, or allow the message to come from a stored file. The file can be a saved message from another system which needs to be forwarded to another system. Incoming messages have to be separated into a local queue and a to-be-forwarded queue. A process had to be designed to periodically check the queue of messages to be forwarded and send on any message found there. The addresses of the messages had to be validated, and the appropriate paths computed for transmission across the network. Some of these tasks require functions and systems calls of a type available on the ONYX/UNIX system but not available on the Honeywell under GCOS without assembly language programming (eg. directory access, clock access, sleeping processes).

Most of these problems have been solved. Some remain for attention in the next project phase. There are also a number of new issues to be addressed, such as dynamic reconfiguration of the network, detection of failed links, broadcast capability, and message logging. The theoretical and mathematical analysis of the system has not been explored. There are database problems to be considered as we integrate the message passing network with the information requirements of the combat support services system. The Georgia Tech project staff is excited at the prospect and ready for the challenge.

Part 3. Software Design Requirements

The generalized system design is shown in figure 1. The modules identified therein are discussed below. Initial versions of those modules have been implemented on all hardware systems used in the AEDs. These initial modules must be generalized and refined to uniformly perform the same functions on each of the systems involved.

All software except the Honeywell dial-out program will be written in a High Level language (Pascal, C, and EDL on the Honeywell, Onyx, and Series/1, respectively). The Honeywell dial-out program will require a small assembly language module to provide modem control.

Path Table

The Path Table will be a two level structure. The first level will contain a list of paths which lead to particular sites (which is dynamically reconfigurable). The second level will contain a table of site names, their telephone numbers (specific to each site), and the name of the local program which will be invoked to establish the connection to the next site in the path.

Sender Program

The Sender Program (which is identical in function for all systems) will be responsible for sending a message to another system. The message will consist of a header, a body, and a special end-of-message signal. The header will contain the path through which the message must be sent (relative to the current system). The body will be the actual text message to be transmitted.

The Sender Program will be invoked after a connection has been completely established with the next system, and will expect to be communicating with a Receiver Program (detailed below) on that system. The message will then be transmitted (path, body, and terminator) via a simple format (line-by-line) to the remote system, using the carriage-return as a line terminator. The end-of-message signal will be the transmission of a special string followed by a carriage return after the last line of valid data in the message body.

The Sender Program will then break the connection with the remote system, after receipt of an acknowledgement (error conditions will be handled later), which will consist of a line count indicating the number of lines received. The Sender Program will then instruct the modem to hang up the phone.

Receiver Program

The Receiver Program will be responsible for receiving messages from a remote system and saving them in a "mailbox" directory, creating a new file for each message received. Messages to be forwarded to another system must be

recognized and queued to be sent by the Sender Program at the end of the session. On the Series/1, such forwarding of messages will not be implemented at this time.

The Receiver Program will be invoked when a connection is established, and will assume that it is communicating with a Sender Program (described above) which is running on the originating system. It must send upon initiation an acknowledgement message to notify the Sender Program that it is currently awaiting a message (header and body). The Receiver Program will maintain a count of the number of lines received from the Sender Program, and will transmit this count to the Sender Program as an acknowledgement upon receipt of the end-of-message signal. The modem should automatically become available when the Sender Program breaks the connection as a result of receiving the acknowledgement.

It will be the responsibility of the Receiver Program at this point to dispose of the message. This means that it must process the header of the message and save the new message (either to be kept on this system or forwarded to another system). Processing the header consists of stripping the current site name from the path that was received in the message header. This revised message (with its modified path) will then be written (header and body) to the new file whose name is derived from a sequence number.

If the new message header is not empty (meaning that this is not the final destination of the message), the Receiver Program will then invoke the Sender Program, indicating to it the name of the file containing the message to be forwarded (its sequence number or file name). The Receiver Program will then terminate, leaving the Sender Program to initiate a transfer of the indicated message to the next site in the network.

Message Generator Program

The message generator program, called GENMSG, will allow an interactive user to type in a text message at his terminal. The program will prompt the user for the destination address and look up the path for that address. The path will be placed in the message header along with the return address. GENMSG will also allow a user to send an existing text file as a message.

Message Display Program

The message display program, called DISPMSG, will allow the interactive user to examine the contents of the message queue. The program will show the message origin and text. The user will be able to browse through the messages, delete messages, or save messages outside the queue. Arriving messages are placed in one of two queues, local or nonlocal. The nonlocal queue is monitored by the Sender program which forwards each message to the next node in its specified path.

Part 4. Advanced Experimental Demonstrations

This part of the report will present detailed descriptions of the demonstrations given during the course of the project. For each demonstration, the purpose, technical issues, and design elements will be identified.

Demonstration 1

The purpose of the first demonstration was to investigate the issues surrounding interprocessor communications between heterogeneous computer systems. The technical issues involved included file transfer mechanisms, positive identification of remote systems, remote console capability, error detection and logging, communications protocols, and automatic establishment of communications via dial-up lines.

The demonstration was conducted using three IBM Series/1 minicomputers. Files were successfully transferred to and from each machine using asynchronous communications links. One Series/1 was connected to a CDC CYBER computer. Files were transferred from the CYBER to the Series/1 and then to another Series/1. The converse was also demonstrated.

Automatic connection to the CYBER was made, with positive identification of remote system and local user. The user at the Series/1 connected to the CYBER and executed console commands interactively, with the Series/1 operating in a pass-through mode. The remote console capability was also demonstrated using two Series/1 computers. All communications were 1200 bits per second asynchronous ASCII protocol.

Most programming was done in the EDL language under the Series/1 Event Driven Executive operating system. In order to get full duplex communications on the Series/1, it was necessary to write a new TTY device driver. The standard TTY driver is half duplex. The new driver was written in Series/1 assembly code.

The demonstration was limited by the lack of availability of 1200 bps modems. The connection between machines were made directly, and arrangement were made for modems to be available for later demonstrations.

Demonstration 2

The purpose of the second demonstration was to investigate remote job initiation and related issues. Issues involved included: remote job initiation, status reporting, output and object routing, automatic routing and disposal, synchronous and asynchronous communications, and interprocessor communications between heterogeneous computer systems.

The three Series/1 computers used in the first demonstration were connected together in a high-speed ring network. One Series/1 was channel attached to

an IBM 4341 super-mini computer. A 9600 bps asynchronous link was made to connect a Series/1 to the CDC CYBER. Another such link was made to a PRIME 550 minicomputer. The third Series/1 was connected to the Ungermann-Bass Net/One local area network via the locally modified Yale ASCII terminal handling system. The 1200 bps modems were not yet available.

The operations described in the first demonstration were repeated on the new network. Additionally, files were transferred to and from the PRIME 550 and the IBM 4341.

An operator at a terminal on the Net/One LAN connected to the IBM 4341 system through the Series/1 Yale ASCII system and initiated jobs and procedures including creation, editing, compilation, and execution of programs. Using a 4341 Host Assembler, a program for the Series/1 was created on the 4341 and compiled, whereupon the object code was transferred automatically down to the Series/1 and loaded. The program listing was routed to another Series/1 which was serving as an automated network print server. This was repeated with the operator at a Series/1 terminal.

The demonstration was limited by the lack of availability of 1200 bps modems. The connections between machines were made directly.

Demonstration 3

The purpose of the third demonstration was to bring together some of the elements of the first two demonstrations and build a simple message transfer system using the machines chosen for the Distributed Combat Service Support system. The TACCS and DAS3 units of the CSS node were to be simulated. Issues addressed included: heterogeneous communications, automatic dial-up of remote systems, file transfer mechanisms, store and forward message passing, error detection and logging, automatic message generation and routing, system security, and fault detection.

A Honeywell DPS Level 6 was used as a DAS3. An ONYX computer running the UNIX operating system was used as a TACCS. An IBM Series/1 computer was also used as a TACCS to demonstrate the ability of the system to work on a number of radically different systems. DC Hayes Smartmodem 1200 modems were used for the TACCS systems. A Cermetek Microelectronics InfoMate modem was used on the DAS3. All connections between machines were made by automatic dial-up over the public telephone network at 1200 bps.

A simple store-and-forward message passing system was demonstrated. Messages generated on each machine were sent to all the other machines. Message destinations were specified as paths through a sequence of network nodes. Messages were generated and transmitted both with and without human intervention. The computer systems automatically initiated and terminated connections through the public telephone network as needed. All traffic through each node was logged for auditing purposes. Transmission errors were detected, and failed message retransmitted. Passwords were employed for the dial-up connections to maintain security. The entire message transmission and

reception process was made transparent to the user.

Close-out Demonstration

The purpose of the final demonstration was to show all facets of t e communications system built for the third demonstration. All issues mentio d above were addressed, and the desired capabilities were demonstrated. e connection of the Honeywell Level 6 (as a DAS3) to the network v s particularly noteworthy.

The system of AED 3 was improved and made more robust. An operating sys m bug on the ONYX system prevented full two-way communication between the O X and the Series/1. The Series/1 could receive messages from the ONYX, but t e ONYX could not receive from both the Series/1 and the Honeywell during e same session. The problem was that the two machines required different par y settings for the ONYX. The UNIX operating system allows one to set a port o disregard incoming parity, but this feature does not work on the ONYX. e chose to set the port so that we could have two-way communications between e Honeywell and the ONYX, and allow one way broadcasts to the Series/1.

Figures 2 through 7 characterize and illustrate the system's operation in t s final demonstration.

Part 5. Recommended Issues for Further Research

The following issues are those which are specific to this project s specified. They show the "tip of the iceberg" of issues related to netwo s of networks of heterogeneous systems.

Failure Detection

A network node should be able to recognize a failed attempt o transmit to another node and take appropriate action, possi y including operator notification and/or automatic re-routing f critical messages. Issues include: failure recognition, scheduli ', network reliability, alternate routing, modem characteristics.

Alternate Routing

Each node in the network which will receive critical messages sho d have one or more defined backup nodes to which messages will e re-routed in the event that the primary destination node is out f service. Issues include: alternate node selection, data redundan ', data consistency, network topology, traffic analysis.

Distributed Database Consistency

If nodes are bypassed while they are out of service, how do we guarantee that their databases are accurate?

Network Traffic Analysis

Identify potential bottlenecks in the network. Estimate the number of ports, modems, and phone lines to be required in a realistic implementation. Develop service time estimates for various classes of messages to be used.

Software Portability

Can the system be moved to other computers? Other operating systems? What changes will be required? What are the machine and system dependencies of such a network?

Error Detection And Correction

How can we guarantee accurate transmission and reception of messages? What methods will provide the greatest security at an acceptable cost in service time and system resources?

Dynamic Network Topology

What will happen to the network if nodes appear and disappear frequently? What if their network addresses change? Can the network be made to grow and adapt without human intervention?

In order to make this a more useful and general system, it will be necessary to provide better error detection and correction measures. Failure detection and message routing must be explored further. It will be advantageous to port the UNIX based system to as many different types of machines as possible to identify any portability problems. The system must be made more robust. The GCOS based part of the system should be translated into C if possible, and the assembly code portions reduced. This should improve portability and make expansion and maintenance easier.

Part 6. System Requirements

The communications system used in the close-out demonstration is limited by certain hardware and software dependencies. This section of the report will define those dependencies.

Operating System Requirements

The system is designed to run under two different operating systems. The TACCS computer will use the UNIX operating system or one of its look-alike systems. The DAS3 is defined to be a Honeywell DPS Level 6 running the GCOS Mod 400 Revision 3.0 operating system.

The software for the UNIX system (the TACCS) is written entirely in the high-level language C. It is not dependent in any way on any particular version of UNIX. It will run without modification under UNIX System III or System V or under Berkeley 4.1 BSD UNIX. During the last month of the project, the UNIX software was ported to a UNIX System III machine and to a Berkeley UNIX (4.1 BSD) system. It was successfully compiled and executed on both systems without change.

The software for the Honeywell Level 6 was written primarily in the high-level language PASCAL, with some special I/O routines done in assembly language. The software is dependent on the GCOS Mod 400 operation system and has not been tested on any other system. Major dependencies are found in the disk and asynchronous I/O routines.

Hardware Requirements

Certain tables used by the software must be set up to indicate the telephone numbers of the network nodes. The system assumes that the modems to be used for dial-out operations are connected to a circuit switched telephone network. The modems must be either Hayes Smartmodem 1200 or Cermetek InfoMate 212A models. The type of modem used for each line must be indicated to the system. These modems are auto-dial, auto-answer, and they return status information during the establishment of a connection.

The modem dependencies have been modularized. Other modems could be used, but would require additional software to support them. The Honeywell requires a modem which does not use an escape series followed by a pause (as required by the Hayes Smartmodem 1200) to indicate the command mode. It is not possible to send a character string followed by a pause from a GCOS PASCAL program without a carriage return character included. This prevents the Hayes modem from being used with the GCOS system. The Cermetek InfoMate meets the requirement. Either modem may be used with the UNIX system.

The computer on which the system is to be run must have at least 1 megabyte of disk space to hold the program system. It must have at least 256K bytes of

RAM to run the system, and one or more user consoles. A printer is useful for logging, but not required. There must be at least one RS232 type asynchronous port on a UNIX system and at least two such ports for a GCOS system.

Part 7. Installation and Operation

This section describes the installation of the system, including all information necessary for maintaining and activating the system at a given site. Since detailed setup varies from machine to machine, this section will detail the steps necessary to manage the system on both the Onyx and the Honeywell DPS-6. It is assumed that these are the only machines on which this system can run.

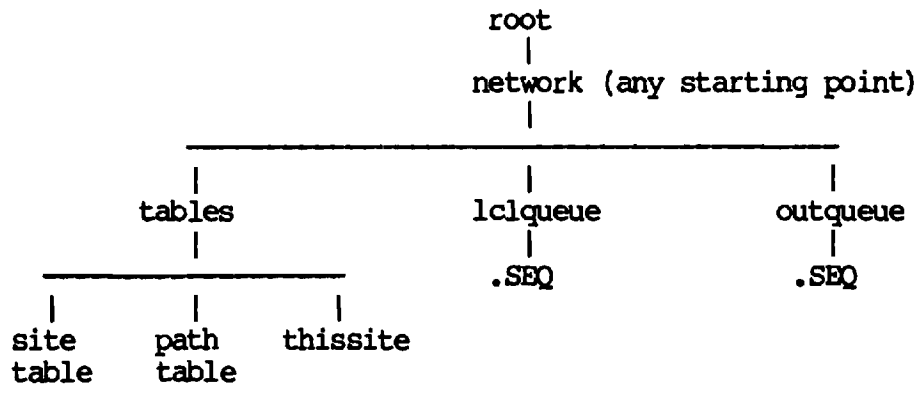
Each section will be divided into two parts, one treating the Onyx, and the other treating the Honeywell. For information on the current configuration and maintenance thereof, refer to the section below, entitled "Current Implementation (Georgia Tech)." To ensure a more thorough understanding of the operating environment of this system, however, it is important that the other sections be read as well.

File Organization and System Configuration

Onyx: The executable code ("xcvr") can reside anywhere on the system, so it is suggested that the file be permitted correctly to all who may need to run the Transceiver (including higher-level directories). This is left to your discretion.

The files that the Transceiver uses, however, must be structured in the following way:

There must be a single directory under which all of the following should reside. Within this top-level directory, there must be three sub-directories: "tables", "lclqueue", and "outqueue." The following is a pictorial representation of this structure:



As illustrated, the "site table", "path table", and "thissite" files are all located in the "tables" sub-directory. The "lclqueue" and "outqueue" sub-directories are empty at the onset, with only a sequence file for each to keep track of incoming files (".SEQ"). These sequence files should contain an

integer (in ASCII) as the first and only element within each file corresponding to the starting sequence number for the files in that sub-directory. (This may be initialized to "0" if desired, when the system is first installed.)

You must now consider the port configuration. There must be a specific single port which is dedicated to the Transceiver. It must not be a standard login port. In other words, do not define this port for login in the system definition for UNIX. (For details on this, consult the UNIX manual.) The port should be defined under "/dev/tty??", where "??" is the port number you want to assign to the Transceiver.

To reflect the configuration of any particular machine, a single source file will need to be modified to reflect the changes. This file is named "net.h", and it accompanies the source files. Edit this file and change all applicable path-names to reflect the hierarchy of the system under construction. This file will contain several lines of definitions similar to the following:

```
#define LCLQUEUE "/v/css/hope/lclqueue"
```

Simply replace these strings to reflect the correct paths according to your definition of the system. Next, change any definitions to reflect the new port address, if it is different from the one defined in the original system. After this is done, recompile the system as described in the section below entitled, "Compiling the Source Code."

The system should now be ready to be activated on this Onyx system.

The program "genmsg" is available on the Onyx only. It can reside anywhere on the system, but must also be recompiled if any changes are made to the file "net.h." This program automatically submits a message typed directly from a user's terminal to the Transceiver for dispatching. Input can be redirected such that a file may be used instead of direct keyboard input. This is accomplished by invoking the program in the following manner: "genmsg < message_file." This message file must match the input normally required by "genmsg", including the first prompt the site to which the message should be sent.

Recompiling "genmsg" is explained in the section "Compiling the Source Code."

Honeywell: There are the same types of files on the Honeywell as there are on the Onyx. The Honeywell components of this system, however, do not all act on the same files, so there is some difference. The executable files are "RECEIVER" and "SENDER." The Sender may be located in any desired directory, but the Receiver must be placed in the directory named ">UDD." For simplest hosting of this system, it would be advisable to match the configuration of your system to that described in the section below, "Current Implementation (Georgia Tech)."

Data File Formats

There are essentially three data files (mentioned above) that are used by the system (on all machines). These are the "Path Table", the "Site Table", and the "Thissite" files. This section will detail the contents of each of these files and the structure of the data represented therein. Note: the file "Thissite" does not exist on the Honeywell.

Path Table: The Path Table is a file (named "paths") which contains a list of all of the available paths to send data from one machine to another possibly through several other systems. This is basically a static routing information file, containing a list of sites and the various sites through which each site can be sent messages. The file is a standard text file, so that adding and deleting information from the file can be performed using any text editor. The structure of this file is as follows:

```
:site                    (ultimate destination)
site!site                (shortest possible path)
site!site!site!site      (alternate path(s))
:site
site!site!site
.
.
```

SiteTable: The Site Table is a file (named "sites") which contains a list of all the direct connections to other sites which are available to this particular machine. It is in essence a list of all the sites in the network that can be accessed directly (either in a hierarchical sense or in actuality), without the need for routing. Along with each site in this file is a list of available phone numbers to that site and a flag indicating the type of connection to be made (either a Honeywell, a Series/1 (defunct), or an Onyx). The structure of the Site Table is described below:

```
:site
type            (integer representing machine type)
phone-number    (phone number relative to this site)
:site
type
phone-number
phone-number
.
.
```

In the above description, "type" is an integer value indicating the type of machine that hosts the referenced site. The values for the type are detailed in the section below, "Current Implementation (Georgia Tech)."

Thissite: This file contains the name of this site as it should be recognized by the other systems in the network. It is, as the others, a standard text file. The only contents are the lower-case letters indicating the name of this site. (To change the name of this site, edit that file and change the name to that of this site. This name must be unique within the network, and cannot be more than 8 characters for this initial version of the system.)

Compiling the Source Code

This section details the steps needed to recompile the source code on either of the two machines currently supported.

Onyx: Change to the source directory on the system (under the current implementation, this directory is "/v/css/hope/oldnet") and type the following commands:

```
$ touch *.c *.h
$ make xcvr
```

(This will display information as it compiles, and will eventually return after having compiled everything. This takes about 15 minutes.)

Once this is done, the executable program "xcvr" will be created in the current directory, which can then be moved to a working directory if desired.

To compile the program "genmsg" in case of changes to the file "net.h", type the following commands in the directory mentioned immediately above:

```
$ touch net.h
```

(this insures that compilation will take place)

```
$ make genmsg
```

Honeywell: Recompilation on the Honeywell is awkward and difficult, and it will not be covered in this guide. There should be no reason to recompile if the system configuration matches the original configuration. Hopefully, it will thus never be necessary to recompile. (At the point that this is being written, some of the original source code has been modified and no concrete explanation of regeneration can be detailed.)

Current Implementation (Georgia Tech)

This section explains the details of the current implementation in the Georgia Tech environment as implemented by the ICS/AIRMICS project during the fiscal year 1983-1984. In this section, it is assumed that only two machines are involved: an Onyx system running the UNIX operating system, and a Honeywell DPS-6 (Level/6) running the GCOS operating system. Since operating system restrictions prevail, some parts of this system are extremely specific to this environment.

For example, on the Honeywell, there must exist an account of the name "GOLDBERG", whose password must be "N360CSC." This can be changed by modifying the source to the Transceiver on the Onyx so that it will not use only that specific account. Such changes should be made to the source file "callho.c" under the source directory "/v/css/hope/oldnet" on what is (at the time of this description) Onyx System D. Recompilation on the Onyx is detailed above ("Compiling the Source Code").

The configuration for the host account on the Honeywell should match the current configurations for the above-mentioned account, and to assure compatibility, the file "START_UP.EC" within that account should also be moved to the host account (assuming you decide to change the account from its current name, "GOLDBERG").

On the Honeywell, there must be two modems, as described previously. One of these must be a Cermetek modem which must be connected to a port named "DIAL06." This same modem will be used for outgoing calls only, invoked by the Sender on the Honeywell when a message is ready to be transmitted. This port ("DIAL06") must not be monitored by the LISTENUR under GCOS. There are system configuration files which must be modified, all detailed in the DPS-6 manual, to ensure that the LISTENUR does not monitor or control the line "DIAL06." The other modem can be connected as a standard dial-in modem, with LISTENUR enabled, since the Onyx will dial in directly and perform a login (to the account "GOLDBERG").

There are three programs of importance on the Honeywell. These are the "SENDER", the "RECEIVER", and "GENMSG." The "RECEIVER" will never be invoked by anyone except the Onyx's Transceiver program, so it is not important but it must reside in the file ">UDD>RECEIVER." The "SENDER" should be invoked by the user when he wishes to transfer a message (contained in ">UDD>MBOX>DRAFT"). The "SENDER" should reside in ">UDD>SENDER." The "GENMSG" program is not completely implemented for the 1983-1984 project, so generating messages must be done through editing the file ">UDD>MBOX>DRAFT" and placing a valid message in that file.

On the Onyx, a modem must be connected for the current configuration to the port "/dev/tty05." This modem must also be a Cermetek, as connected to the Honeywell, described above. It will be used for both sending and receiving, so for the purposes of this software, it could be the only modem on the system. (This, of course, is no requirement.)

The Onyx program is called "xcvr", and must be invoked before any messages can be transacted between the two machines. It will run until terminated by the operator, servicing requests for both transmission and reception of messages over the modem. This program is normally in a "debug" state, echoing various sorts of information to the terminal from which the program was initially invoked. If you do not wish to see this status information, you may do one of two things: change a flag and recompile the source, or reroute all output to /dev/null. The first may be accomplished by editing the file "net.h" under the source directory and deleting the line "#DEFINE XDEBUG." Then the source will need to be recompiled as detailed in the section above, "Compiling the Source Code." The second, simpler method, is to type "xcvr > /dev/null", which will reroute all standard-error messages to the null device. There is a second type of message that will appear on this "operator's terminal." This is the "informative message", indicating that a transfer has taken place. This is usually important, and thus worth seeing.

On the Onyx, all transactions and important status indicators are maintained in a sequential "log file." This log file resides in "/v/css/hope/log/logfile" and may be examined at any time by any user. The purpose of this file is simply to maintain a permanent copy of error messages and informative status messages for retrospection or possible debugging. For a blow-by-blow status report, you can always type "tail -f /v/css/hope/log/logfile" on another terminal (not the one you loaded "xcvr" on, since it is occupied). This will list the last changes to the file as they appear, so you can watch new developments, assuming the Transceiver is activated and transactions are taking place.

For the most part, the Onyx software will recover from errors. If it cannot, the Transceiver will completely terminate, causing a need for inspection of the problem, possible repair, and re-execution of the program "xcvr." On the other hand, the Honeywell is less able to cope with such problems as I/O hangs, etc. If the Honeywell fails to participate in its half of the conversation, use your own judgement in relieving the problem. If nothing else, abort the program that may be hung and try to start again (the Onyx will reset and also try again automatically in most cases). If it is impossible to kill the program on the Honeywell, reset the Honeywell altogether. This is sometimes the only way to deal with intermittent Honeywell failures.

On both machines, the "Site Table" file contains codes identifying the type of connections made to another site. Currently, a type of "1" means that the remote machine is a Series/1, a type of "4" means that the remote machine is an Onyx, and a type of "9" means that the remote machine is a Honeywell.

This information should be sufficient to maintain the system as it is currently implemented in the Georgia Tech environment. If it is not, feel free to contact either Steven Goldberg or Bill Putnam in the ICS/AIRMICS project office at (404) 894-4311.

Part 8

Figures 1-7

Message System Design

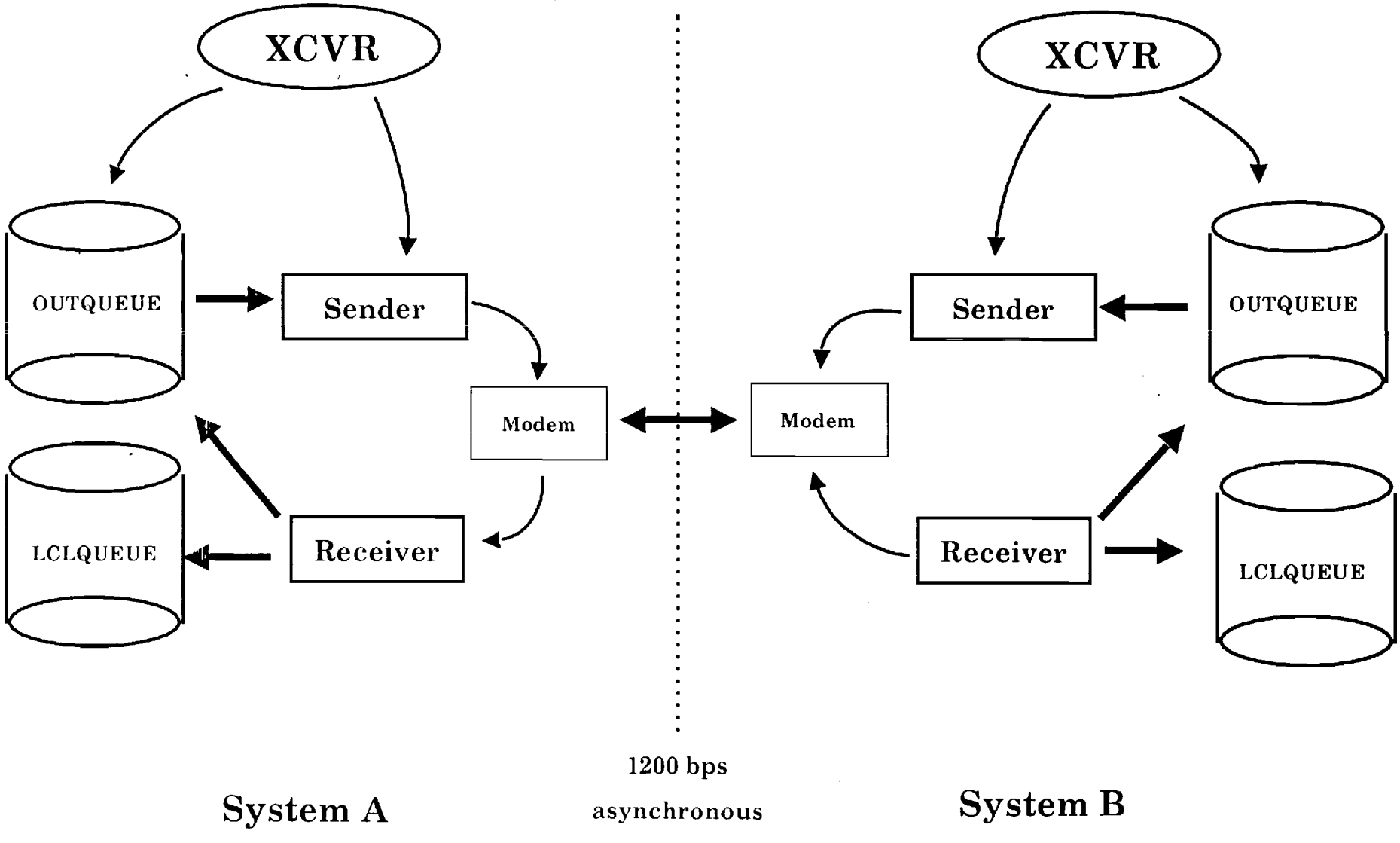
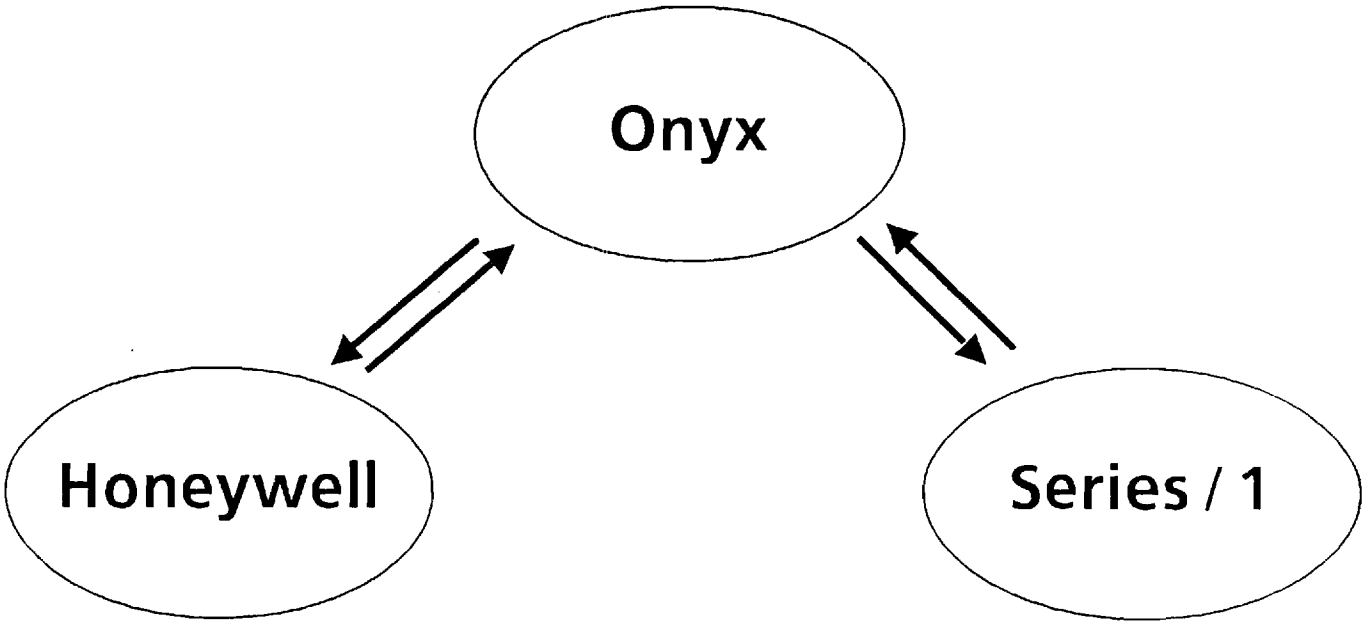


Figure 1

System Configuration



Machine-specific attributes

Onyx:
Unix, one bidirectional phone line

Honeywell:
GCOS-6, two unidirectional phone lines

Series / 1:
EDX, one bidirectional phone line

Figure 2

Tables

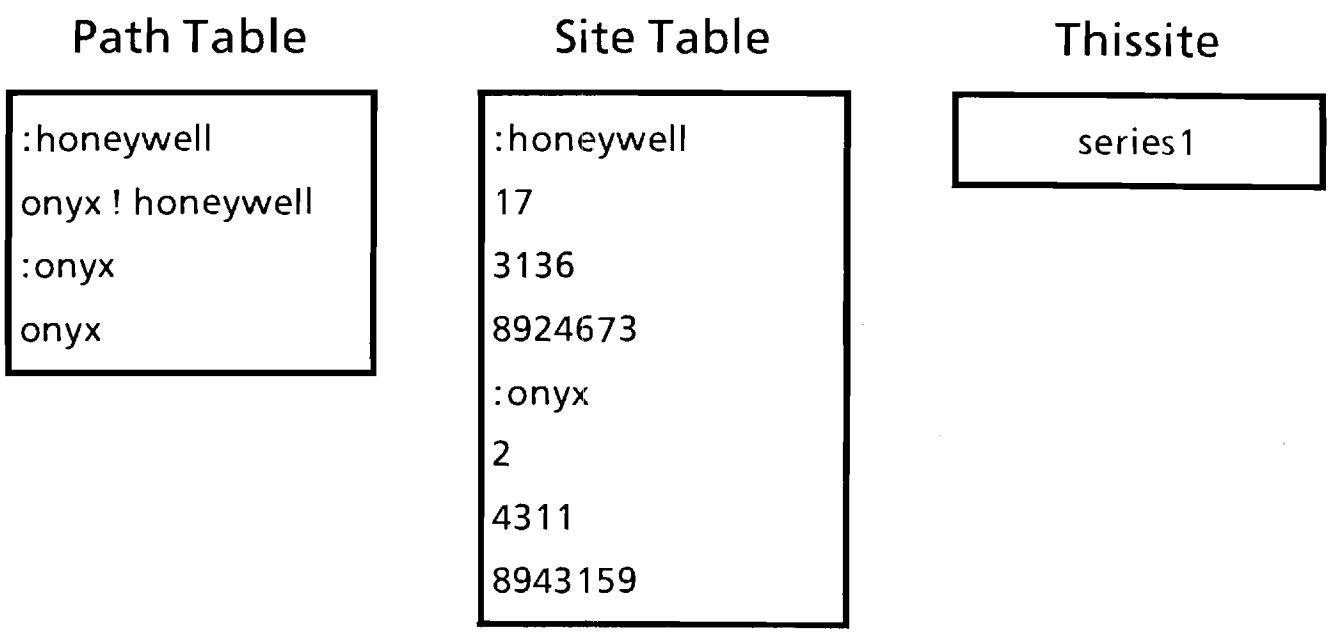
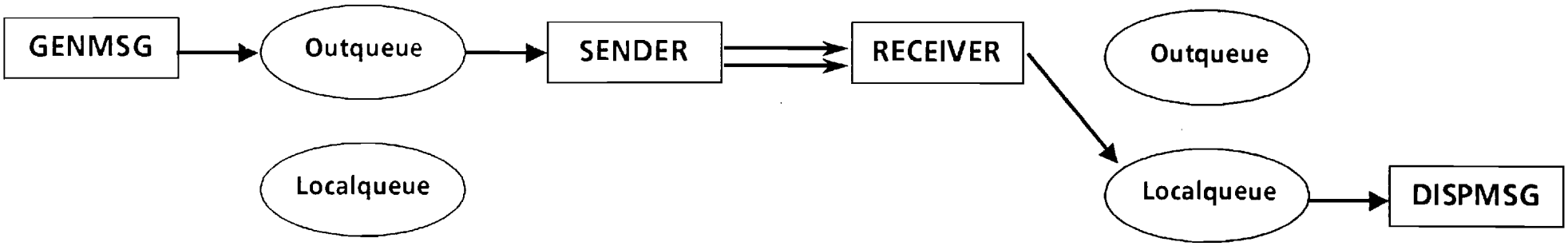


Figure 3

Generate & Display:



Generate & Forward:

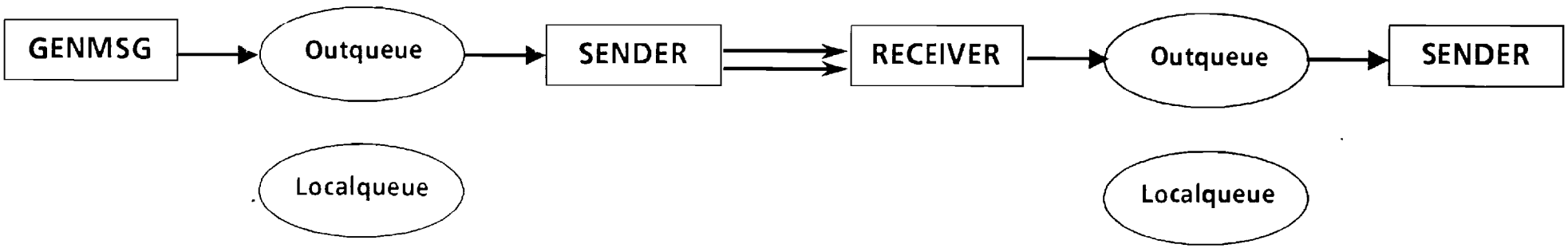


Figure 4

Data Flow for GENMSG

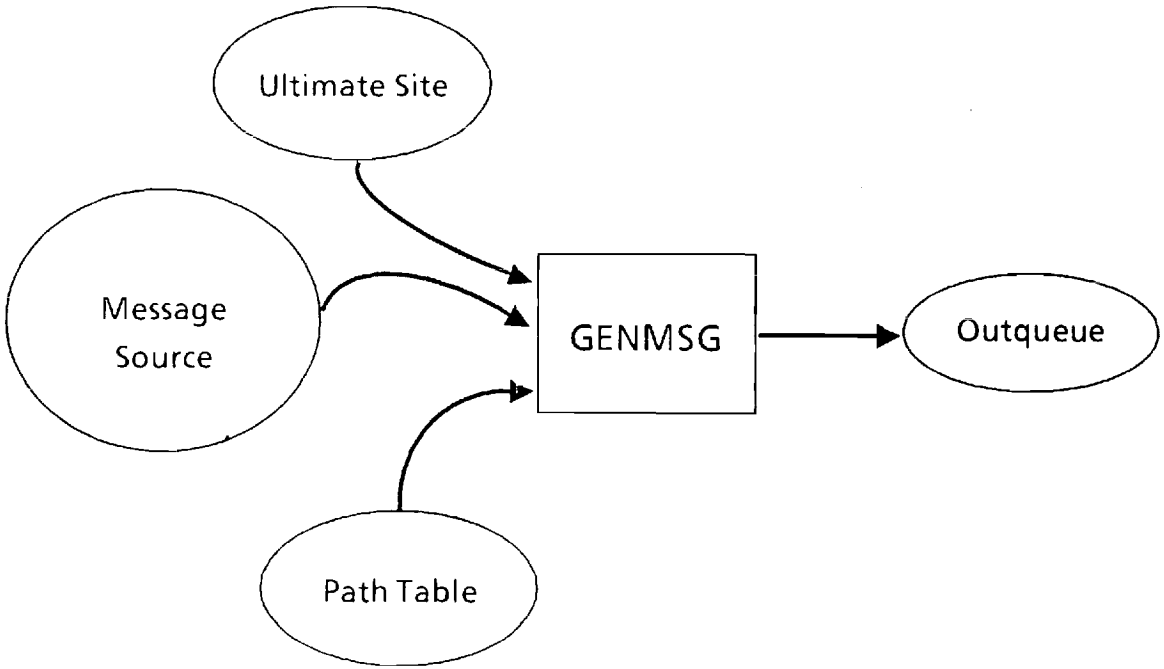


Figure 5

Data Flow for SENDER

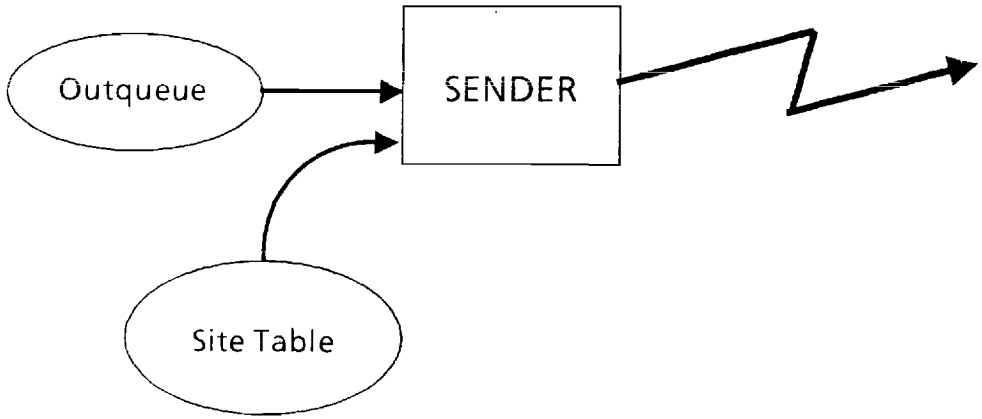


Figure 6

Data Flow for RECEIVER

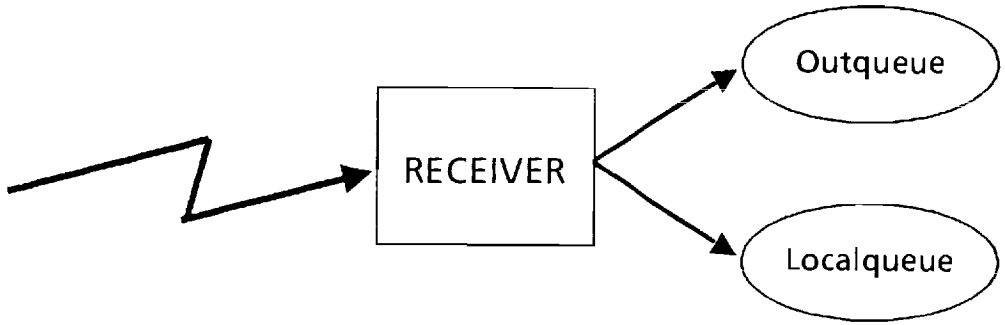


Figure 7

Part 9

Program Source Listings

UNIX System Source Code

C programs

Table of Contents Wed Nov 14 16:53:33 1984

```
clean:
	rm -f *.o

read.o: net.h read.c
	cc -O -c read.c

readdir.o: net.h readdir.c
	cc -O -c readdir.c

firstfile.o: net.h firstfile.c
	cc -O -c firstfile.c

dial.o: net.h dial.c
	cc -O -c dial.c

callho.o: net.h callho.c
	cc -O -c callho.c

callsl.o: net.h callsl.c
	cc -O -c callsl.c

annoy.o: net.h annoy.c
	cc -O -c annoy.c

datetime.o: net.h datetime.c
	cc -O -c datetime.c

strsave.o: strsave.c
	cc -O -c strsave.c

frename.o: net.h frename.c
	cc -O -c frename.c

genmsg: net.h getseqno.o validpath.o myname.o frename.o
	cc -O -o genmsg genmsg.c getseqno.o validpath.o myname.o frename.o
	chmod a+x genmsg

getseqno.o: net.h getseqno.c
	cc -O -c getseqno.c

handlelogin.o: net.h handlelogin.c
	cc -O -c handlelogin.c

hangup.o: net.h hangup.c
	cc -O -c hangup.c

myname.o: net.h myname.c
	cc -O -c myname.c

openmodem.o: net.h openmodem.c
	cc -O -c openmodem.c

readstr.o: net.h readstr.c
	cc -O -c readstr.c

receiver.o: net.h receiver.c
	cc -O -c receiver.c

sender.o: net.h sender.c
	cc -O -c sender.c
```

```
stripme.o: net.h stripme.c
	cc -O -c stripme.c

validpath.o: net.h validpath.c
	cc -O -c validpath.c

validsite.o: net.h validsite.c
	cc -O -c validsite.c

writelog.o: net.h writelog.c
	cc -O -c writelog.c

xcvr.o: net.h xcvr.c
	cc -O -c xcvr.c

xcvr: xcvr.o openmodem.o sender.o receiver.o handlelogin.o\
    read.o readdir.o dial.o callho.o callsl.o getseqno.o writelog.o annoy.o\
    handlelogin.o myname.o openmodem.o readstr.o receiver.o datetime.o\
    sender.o stripme.o validpath.o validsite.o hangup.o frename.o strsave.o\
    firstfile.o
	cc -O -o xcvr xcvr.o readdir.o dial.o callho.o callsl.o getseqno.o\
    writelog.o handlelogin.o myname.o openmodem.o readstr.o receiver.o\
    datetime.o annoy.o sender.o stripme.o validpath.o validsite.o\
    hangup.o read.o frename.o strsave.o firstfile.o
	chmod a+x xcvr

disp.o: net.h disp.c
	cc -O -c disp.c

disp: disp.o strsave.o
	cc -O -o disp disp.o strsave.o readdir.o stripme.o
	chmod a+x disp
```

```
#include "net.h"

int Annoy (P1, P2, P3, P4)

char *P1, *P2, *P3, *P4;

/* For now, writes a message to the operator.  In the future, it will */
/* probably send mail to someone or write to some other logfile.      */

{
   char Now [26];

   DateTime (Now);

   fprintf (stderr, "annoy: %s %s %s %s %s\n", Now, P1, P2, P3, P4);

   return (0);

}
```

```
#define KLUDGE


#include "net.h"


char LoginPrompt [] = "LOGIN  ";
char LoginString [] = "L GOLDBERG\r";
char Password [] = "N360CSC\r";
char LoadString [] = "RECEIVER\r";

int CallHo (ModemFd)

int ModemFd;

{
   struct   termio TTYSet;
   char ReadChr ();
   char AckString [20];

   register int Count;
   int PromptLength = strlen (LoginPrompt);

   /* Set up the modem for Honeywell's parity */

   if (ioctl (ModemFd, TCGETA, &TTYSet) == ERR)
      {
       fprintf (stderr, "CallHo: Can't get old modem settings\n");
       exit (1);
      }

/*
   TTYSet.c_cflag &= ~CSIZE;
   TTYSet.c_cflag |= (PARENB | CS7);
*/
   TTYSet.c_oflag &= ~CRDLY;
   TTYSet.c_oflag |= (CR3 | OPOST);

   if (ioctl (ModemFd, TCSETA, &TTYSet) == ERR)
      {
       fprintf (stderr, "CallHo: Can't set new modem settings\n");
       exit (1);
      }

   sleep (2);

   write (ModemFd, CR, 1);

   do {
       Count = 0;
       while (ReadChr (ModemFd) == LoginPrompt [Count])
          Count++;
      } while (Count < PromptLength);
/* #ifdef XDEBUG */
fprintf (stderr, "Login string detected. Attempting login.\n");
/* #endif */

   /* Found login prompt.  Sleep for a few seconds and flush input buffer. */

   sleep (3);
```

```
   FlushModemInput (ModemFd);

   /* Log in */

   write (ModemFd, LoginString, strlen (LoginString));
#ifdef XDEBUG
fprintf (stderr, "%s", LoginString);
#endif

   sleep (2);  /* Wait for Password prompt */

   write (ModemFd, Password, strlen (Password));

   sleep (20);  /* Wait to log all the way in */     /* KLUDGE */

   write (ModemFd, LoadString, strlen (LoadString));

#ifdef KLUDGE
sleep (2);
goto kludge;
#endif
   while (!EQUALS (ReadStr (ModemFd, AckString), ACKMSG))
         {
#ifdef XDEBUG
          fprintf (stderr, "CallHo: Waiting for ACK.\n");
#endif
          if (AckString == NULL)
             fprintf (stderr, "CallHo: Timed out.\n");
#ifdef XDEBUG
          else
             fprintf (stderr, "CallHo: Received '%s'\n", AckString);
#endif
         }
#ifdef KLUDGE
kludge:
#endif

   return (0);
}
```

```
#define MAXRETRY 20

char ReadChr (Fd)

int Fd;

{
   register char Ch;
   register int  NumTries = 0;

   while ((Read (Fd, &Ch, 1) == 0) && (NumTries < MAXRETRY))
```

```
        {
         NumTries++;
#ifdef XDEBUG
         fprintf (stderr, "ReadChr: Timed out on read.\n");
#endif
         write (Fd, CR, 1);
        }

    if (NumTries >= MAXRETRY)
       {
        fprintf (stderr, "ReadChr: The Honeywell is not there.\n");
        WriteLog ("CallHo:", "Honeywell appears down or disconnected", "", "");
        return (ERR);
       }

    return (Ch);
}
```

```
#include "net.h"

char LoadMessage [] = "$L #RECEIVE\r";

int CallS1 (ModemFd)

int ModemFd;

{
   struct   termio TTYSet;

   char AckString [128];
   char *RetVal;
   register int Count;

  . /* Set up the modem for Series/1 */

   if (ioctl (ModemFd, TCGETA, &TTYSet) == ERR)
      {
       fprintf (stderr, "CallS1: Can't get old modem settings\n");
       exit (1);
      }

   TTYSet.c_cflag &= ~CSIZE;
   TTYSet.c_cflag |= CS8;
   TTYSet.c_cflag &= ~PARENB;
   TTYSet.c_oflag &= ~CRDLY;
   TTYSet.c_oflag |= (CR3 | OPOST);

   if (ioctl (ModemFd, TCSETA, &TTYSet) == ERR)
      {
       fprintf (stderr, "CallS1: Can't set new modem settings\n");
       exit (1);
      }

   sleep (3);

   FlushModemInput (ModemFd);

   /* Load the receiver program */

   write (ModemFd, "\033", 1);

   sleep (1);

   write (ModemFd, LoadMessage, strlen (LoadMessage));

   do {
       RetVal = ReadStr (ModemFd, AckString);
#ifdef XDEBUG
       fprintf (stderr, "CallS1: Waiting for ACK.\n");
#endif
       if (RetVal == NULL)
           fprintf (stderr, "CallS1: Timed out.\n");
#ifdef XDEBUG
       else
           fprintf (stderr, "CallS1: Received '%s'\n", RetVal);
#endif
      } while (!EQUALS (AckString, ACKMSG));
```

```
    return (0);
}
```

```
#include "net.h"

int DateTime (Str)

char *Str;

{
   long BDate;

   BDate = time ((long *) 0);
   sprintf (Str, "%s", ctime (&BDate));
   Str [strlen (Str)-1] = '\0';          /* Zap NL placed by TIME (2) */

   return (0);

}
```

```
#include "net.h"

#define MAXRETRY 20

int Dial (ModemFd, DialProg, TelNums)

int  ModemFd;
int  DialProg;
char *TelNums [20];

{
   char DialCmd [40];
   char FromModem [20];
   char Answer [2];
   struct termio TTYSet;
   int  Oflag, Cflag;
   register int RetVal, NumTries=0;

/* Using Hayes Modem - Minimal Code */

   if (ioctl (ModemFd, TCGETA, &TTYSet) == ERR)
      {
       fprintf (stderr, "Dial: Can't get old modem settings\n");
       exit (1);
      }

   Oflag = TTYSet.c_oflag;
   Cflag = TTYSet.c_cflag;
   TTYSet.c_oflag |= (CR3 | OPOST);

   if (ioctl (ModemFd, TCSETA, &TTYSet) == ERR)
      {
       fprintf (stderr, "Dial: Can't set new modem settings\n");
       exit (1);
      }

   write (ModemFd, CR, 1);

   sleep (1);

   FlushModemInput (ModemFd);

   sprintf (DialCmd, "ATD%s\r", TelNums[0]);
   write (ModemFd, DialCmd, strlen (DialCmd));
   sleep (2);

   FlushModemInput (ModemFd);

   sleep (4);  /* Give modem minimum time to dial & connect */

   do {
       RetVal = Read (ModemFd, Answer, 2);
       NumTries++;
      } while ((RetVal == 0) && (NumTries < MAXRETRY));

   if (RetVal == 0)
      {
       WriteLog ("DialHayes: Modem hung on initial dial command", "", "", "");
       return (ERR);
      }
```

```
   TTYSet.c_oflag = Oflag;

   if (ioctl (ModemFd, TCSETA, &TTYSet) == ERR)
      {
       fprintf (stderr, "Dial: Can't reset modem settings\n");
       exit (1);
      }


   if (Answer[0] != '1')
      {
       fprintf (stderr, "DialHayes: Modem answered %c\n", Answer[0]);
       HangUp (ModemFd, 0);
       return (ERR);
      }


   switch (DialProg)
      {
       case 1 : /* Series/1 - invokes receiver directly */
                return (CallS1 (ModemFd));
       case 4 : /* Onyx - uses Transceiver */
#ifdef XDEBUG
                fprintf (stderr, "\nDial: Connected, about to send LOGINMSG\n");
#endif
                sleep (3);
                FlushModemInput (ModemFd);
                write (ModemFd, "     \r", 6);
                write (ModemFd, LOGINMSG, strlen (LOGINMSG));
                write (ModemFd, CR, 1);
                while (!EQUALS (ReadStr (ModemFd, FromModem), ACKMSG))
                   {
#ifdef XDEBUG
                    fprintf (stderr, "Dial: waiting for ack\n");
#endif
                   }
                return (0);
                break;
       case 9 : /* Honeywell - logs in and invokes receiver */
                return (CallHo (ModemFd));
       default: /* Unidentified receiving machine */
                fprintf (stderr, "Dial: Don't know dial type %d\n", DialProg);
                return (ERR);
                break;
      }
}
```

```
/* This program MUST be run with "sh -c disp", or getenv ("PWD") won't work */

#include "net.h"

#define ALIVE  0
#define DELETE 1
#define COPY   0
#define MOVE   1

struct Msg
   {
    char  *FileName;
    int   MsgNum;
    short Status;
    char  *Orig;
   }









     *Msg [128];

int  NumMsgs;
char *OrigCwd;
char Comm;
char Queue [48];
char Parm1 [30];
char Parm2 [30];
int NumParms;


main (argc, argv)

int argc;
char **argv;

{
   switch (argc)
      {
       case 1:
                strcpy (Queue, LCLQUEUE);
                break;
       case 2:
                strcpy (Queue, argv [1]);
                break;
       default:
                Usage ();
      }

   Startup ();
   Headers ();

   while (1)
      {
       Comm = GetComm ();
```

```
        switch (Comm)
           {
            case 'T': Type ();          break;

            case 'D': Delete (DELETE); break;

            case 'C': Copy (COPY);      break;

            case 'M': Copy (MOVE);      break;

            case 'H': Headers ();       break;

            case 'Q': Quit ();          break;

            case 'U': Delete (ALIVE);   break;

            case '?': Help ();          break;

            default:  Error ();         break;
           }
      }
}



int Startup ()

/*
   Gets the message filenames and sets up the Msg structs.
*/

{
   register int i = 0;
   char Junk;
   char *FileNames [128];

   OrigCwd = StrSave (getenv ("PWD"));     /* Save original directory */

   if ((chdir (Queue) == ERR)||((NumMsgs = ReadDir (Queue, FileNames)) == ERR))
      {
       fprintf (stderr, "There's no way I can read that directory!\n");
       exit (1);
      }

   for ( ; i < NumMsgs; i++)
       {
        Msg [i] = (struct Msg *) calloc (1, sizeof (struct Msg));
        Msg [i]->FileName = StrSave (FileNames [i]);
        sscanf (FileNames [i], "%c%5d", &Junk, &(Msg [i]->MsgNum));
        Msg [i]->Status = ALIVE;
        DoFileStuff (Msg [i]);
       }
```

```
    if (chdir (OrigCwd) == ERR)
        fprintf (stderr, "Disp: Can't chdir\n");

}
```

```
int DoFileStuff (Msg)

struct Msg *Msg;

/*
   Opens the Msg file, gets the Orig field, and sets Status accordingly.
*/

{
   char TempStr [20];
   register FILE *Fd;

   if ((Fd = fopen (Msg->FileName, "r")) == NULL)
      {
       fprintf (stderr, "Disp: Can't open %s\n", Msg->FileName);
       Msg->Status = ERR;
      }
   else
      {
       SkipEOL (Fd);                    /* Skip first line */
       fscanf (Fd, "%s", TempStr);
       Msg->Orig = malloc (20);
       StripMe (TempStr, Msg->Orig);    /* Orig = First name in TempStr */
       fclose (Fd);
      }
}
```

```
int GetComm ()

{
   char TempStr [40];

   do {
       Comm = ' ';
       fputs ("Disp> ", stdout);
       if (fgets (TempStr, 40, stdin) == NULL)
```

```
            Quit ();
         TempStr [strlen (TempStr) - 1] = '\0';
         NumParms = sscanf (TempStr, "%c %s %s", &Comm, Parm1, Parm2);
        } while ((NumParms < 2) && (Comm == ' '));

   return (islower (Comm) ? _toupper (Comm) : Comm);
}



Error ()

{
   puts ("I can't understand you. Please type '?' for help.");
}



Help ()

{
   puts ("c M# file   COPY Msg number M# to file");
   puts ("d M#        DELETE Msg number M#");
   puts ("h           Display message HEADERS");
   puts ("m M# file   MOVE Msg number M# to file");
   puts ("q           QUIT program");
   puts ("t M#        TYPE Msg number M#");
   puts ("u M#        UNDELETE Msg Number M#");
}



Type ()

{
   register int i = 0;
   register int Ch;
```

```
    register int Num;
    int Ok = FALSE;
    register FILE *Fd;

    if (NumParms != 2)
        {
         Error ();
         return;
        }

    Num = atoi (Parm1);

    for (i = 0; i < NumMsgs; i++)
         if (Num == Msg [i]->MsgNum)
             {
              Ok = TRUE;
              break;
             }

    if (!Ok)
        {
         printf ("Message number %d doesn't exist.\n", Num);
         return;
        }


    Num = i;

    if (chdir (Queue) == ERR)
        {
         fprintf (stderr, "Disp: Can't chdir\n");
         return;
        }

    if ((Fd = fopen (Msg [Num]->FileName, "r")) == NULL)
        {
         fprintf (stderr, "Disp: Can't open %s\n", Msg [Num]->FileName);
         Msg [Num]->Status = ERR;
         return (0);
        }

    printf ("Message %d\n", Msg [Num]->MsgNum);
    printf ("From: %s\n", Msg [Num]->Orig);
    fseek (Fd, 0L, 0);
    SkipEOL (Fd);
    printf ("Path: ");
    while ((Ch = getc (Fd)) != EOF)
           putc (Ch, stdout);
    putc ('\n', stdout);
    fclose (Fd);

    if (chdir (OrigCwd) == ERR)
        fprintf (stderr, "Disp: Can't chdir\n");

}
```

```
Headers ()

{
   register int i = 0;

   if (NumMsgs == 0)
       return;

   puts ("\nNum  From");

   for ( ; i < NumMsgs; i++)
       {
        printf ("%3d  %-20s", Msg [i]->MsgNum, Msg [i]->Orig);
        puts ((Msg [i]->Status == DELETE) ? " (deleted)" : "");
       }

   putc ('\n', stdout);
}
```

```
Delete (Mode)

int Mode;

{

   register int Num;
   register int i = 0;
   int Ok = FALSE;

   if (NumParms != 2)
      {
       Error ();
       return;
      }

   Num = atoi (Parm1);

   for ( ; i < NumMsgs; i++)
        if (Num == Msg [i]->MsgNum)
           {
            Ok = TRUE;
            break;
           }

   if (Ok)
       Msg [i]->Status = Mode;
   else
       printf ("Message number %d doesn't exist.\n", Num);
```

```
}
```

```
Copy (Mode)

int Mode;    /* COPY or MOVE */

{
   register int Num;
   register int i = 0;
   int Ok = FALSE;
   FILE *Fd1, *Fd2;
   register char Ch;

   if (NumParms != 3)
      {
       Error ();
       return;
      }

   Num = atoi (Parm1);

   for ( ; i < NumMsgs; i++)
        if (Num == Msg [i]->MsgNum)
           {
            Ok = TRUE;
            break;
           }

   if (!Ok)
      {
       printf ("Message number %d doesn't exist.\n", Num);
       return;
      }

   if ((Fd1 = fopen (Parm2, "w")) == NULL)
      {
       printf ("I'm sorry, but I can't open %s for writing.\n", Parm2);
       return;
      }

   if (chdir (Queue) == ERR)
      {
       fprintf (stderr, "Disp: Can't chdir\n");
       return;
      }

   if ((Fd2 = fopen (Msg [i]->FileName, "r")) == NULL)
      {
       fprintf (stderr, "Disp: Can't open %s\n", Msg [i]->FileName);
       return;
```

```
        }

    while ((Ch = getc (Fd2)) != EOF)
            putc (Ch, Fd1);

    fclose (Fd1);
    fclose (Fd2);

    if (Mode == MOVE)
        Msg [i]->Status = DELETE;

    if (chdir (OrigCwd) == ERR)
        fprintf (stderr, "Disp: Can't chdir\n");
}




Quit ()

{
    register int i = 0;

    if (chdir (Queue) == ERR)
        {
        fprintf (stderr, "Disp: Can't chdir\n");
        exit (1);
        }

    for ( ; i < NumMsgs; i++)
         if (Msg [i]->Status == DELETE)
             if (unlink (Msg [i]->FileName) == ERR)
                 fprintf (stderr, "Disp: Can't unlink %s\n", Msg [i]->FileName);

    puts ("Have a nice day!\n");

    exit (0);
}




Usage ()

{
    fprintf (stderr, "Usage: disp [queue]\n");
    exit (1);
```

```
}
#include "net.h"

char *FirstFile (Dir)

char *Dir;

/*
   Returns a pointer to the first file in Dir returned by ReadDir,
   or a NULL pointer if no files exist.
*/

{
   char *Files [128];

   if (ReadDir (Dir, Files) > 0)
       return (StrSave (Files [0]));
   else
       return ((char *) NULL);
}
```

```
#include "net.h"

int FRename (Path1, Path2)

char *Path1;
char *Path2;

/* Renames Path1 to Path2 */

{
   if (link (Path1, Path2) == ERR)
       return (ERR);

   if (unlink (Path1) == ERR)
       return (ERR);

   return (0);
}
```

```
#include "net.h"

main ()

{
   FILE *TmpFileFd;
   char TmpFileName [128];
   char MsgFileName [128];
   char SiteName [128];
   char DestPath [128];
   register int Ch;
   int  SeqNo;


   printf ("To: ");
   gets (SiteName);

   if (!ValidPath (SiteName, DestPath))     /* Leaves path in DestPath */
      {
       fprintf (stderr, "%s: Unknown destination.\n", SiteName);
       exit (1);
      }

   SeqNo = GetSeqNo (OUTQUEUE);

   sprintf (TmpFileName, "%s/.M%.5d", OUTQUEUE, SeqNo);
   sprintf (MsgFileName, "%s/M%.5d", OUTQUEUE, SeqNo);

   if ((TmpFileFd = fopen (TmpFileName, "w")) == NULL)
      {
       fprintf (stderr, "GenMsg: Can't create %s\n", TmpFileName);
       exit (1);
      }

   fprintf (TmpFileFd, "%s\n%s\!%s\n", DestPath, MyName (), DestPath);

   printf ("Enter message, end with a control-D.\n");

   while ((Ch = getc (stdin)) != EOF)
         {
          putc (Ch, TmpFileFd);
         }

   fclose (TmpFileFd);

   if (FRename (TmpFileName, MsgFileName) == ERR)
      {
       fprintf (stderr, "GenMsg: Can't rename !!!\n");
       exit (1);
      }

   if (chmod (MsgFileName, 0666) == ERR)
      {
       fprintf (stderr, "GenMsg: Can't chmod !!!\n");
       unlink (MsgFileName);
       exit (1);
      }
}
```

```
#include "net.h"

int GetSeqNo (DirName)

char *DirName;

/* Returns an available sequence number for a file in the given DirName. */

{
   int  SeqNo;
   FILE *SeqFileFd;
   char SeqFileName [40];

   sprintf (SeqFileName, "%s/.SEQ", DirName);

   if ((SeqFileFd = fopen (SeqFileName, "r+")) == NULL)
      {
       fprintf (stderr, "GetSeqNo: Can't open %s\n", SeqFileName);
       exit (1);
      }

   fscanf (SeqFileFd, "%d", &SeqNo);

   if (SeqNo < MAXSEQNO)
       SeqNo++;
   else
       SeqNo = 1;

   fseek (SeqFileFd, 0L, 0);
   fprintf (SeqFileFd, "%d       ", SeqNo);
   fclose (SeqFileFd);

   return (SeqNo);
}
```

```
#include "net.h"


char LoginCommand [] = LOGINMSG;

int HandleLogin (Fd)

int Fd;

/* Validates call-in connection from remote sender.      */
/* Returns 1 if validated.                               */

{
   char Ch, LoginLine [8];
   int  LoginStatus;
   int  LoginLength = strlen (LoginCommand);
   register int NumChars;
   register int Count, NumTries = 0;

   do {
      Count = Read (Fd, &Ch, 1);
#ifdef XDEBUG
fprintf (stderr, "Handlelogin: Scanning for CR. ('%.2x')\n", Ch);
#endif
     } while ((Ch != '\r') && (Count != 0));

   if (Count == 0)
      {
#ifdef XDEBUG
       fprintf (stderr, "HandleLogin: Timed out on initial read.\n");
#endif
       return (FALSE);
      }

#ifdef XDEBUG
fprintf (stderr, "HandleLogin: Found CR, scanning for 'LOGIN'\n");
#endif
   do {
       Count = 0;
       NumChars = Read (Fd, &Ch, 1);
       while ((Ch == LoginCommand [Count]) && (NumChars != 0))
          {
           Count++;
           Read (Fd, &Ch, 1);
          }
       if (NumChars == 0)
          {
           fprintf (stderr, "HandleLogin: Timed out reading login message.\n");
           return (FALSE);
          }
#ifdef XDEBUG
fprintf (stderr, "HandleLogin: end of scan loop.\n");
#endif
      } while (Count < LoginLength);
#ifdef XDEBUG
fprintf (stderr, "Login string detected.\n");
#endif

   write (Fd, ACKMSG, strlen (ACKMSG));
```

```
    write (Fd, CR, 1); /* Xmit Carriage-Return */

    return (TRUE);
}
```

```
#include "net.h"

#define HONEYWELL 9      /* Later - put this in net.h */
#define SERIES1 1

int HangUp (Fd, DialProg)

int Fd;
int DialProg;

/*
   Sends a hangup command to the modem
*/

{
   struct termio TTYSet;

   if (DialProg == HONEYWELL)
      {
       sleep (5);                /* Ensure that the receiver has ended */
       write (Fd, "BYE\r", 4);   /* Transmit logoff request */

       if (ioctl (Fd, TCGETA, &TTYSet) == ERR)
          {
           fprintf (stderr, "HangUp: Can't get old modem settings\n");
           exit (1);
          }

       TTYSet.c_oflag &= ~(CR3 | OPOST);

       if (ioctl (Fd, TCSETA, &TTYSet) == ERR)
          {
           fprintf (stderr, "HangUp: Can't set new modem settings\n");
           exit (1);
          }

       sleep (4);
      }

   if (DialProg == SERIES1)
      {
       if (ioctl (Fd, TCGETA, &TTYSet) == ERR)
          {
           fprintf (stderr, "HangUp: Can't get old modem settings\n");
           exit (1);
          }

       TTYSet.c_cflag &= ~CSIZE;
       TTYSet.c_cflag |= (PARENB | CS7);
       TTYSet.c_oflag &= ~(CR3 | OPOST);

       if (ioctl (Fd, TCSETA, &TTYSet) == ERR)
          {
           fprintf (stderr, "HangUp: Can't set new modem settings\n");
           exit (1);
          }
      }
```

```
    sleep (2);
    write (Fd, "+++", 3);     /* For Hayes ONLY */
    sleep (2);
    write (Fd, "ATH\r", 4);   /* For Hayes ONLY */
    sleep (2);
    FlushModemInput (Fd);     /* Flush input buffer */
    return (0);
}
```

```
#include "net.h"

char *MyName ()

{
   char *RetPtr;
   FILE *Fd;

   RetPtr = malloc (20);

   if ((Fd = fopen (THISSITEFILE, "r")) == NULL)
      {
       fprintf (stderr, "MyName: Can't open %s\n", THISSITEFILE);
       exit (1);
      }_

   fscanf (Fd, "%s", RetPtr);

   fclose (Fd);

   return (RetPtr);
}
```

```
#define XDEBUG


#include <stdio.h>
#include <termio.h>
#include <fcntl.h>
#include <ctype.h>
#include <sys/types.h>


#define TRUE 1
#define FALSE !TRUE
#define EQUALS !strcmp
#define SkipEOL(fd) while(getc(fd)!='\n');
#define FlushModemInput(fd) ioctl(fd,TCFLSH,0)

#define MODEMLINE "/dev/tty05"

#define LOGINMSG "LOGIN"
#define ACKMSG "ACK"
#define NACKMSG "NACK"

#define EOMSG "?MATZOH?"
#define SEPCHAR '!'

#define CR "\015"
#define ERR -1
#define MAXTRIES 700

#define LCLQUEUE "/v/css/hope/lclqueue"
#define OUTQUEUE "/v/css/hope/outqueue"
#define MAXSEQNO 32767

#define PATHTABLEFILE "/v/css/hope/tables/paths"
#define SITETABLEFILE "/v/css/hope/tables/sites"
#define THISSITEFILE  "/v/css/hope/tables/thissite"
#define LOGFILE       "/v/css/hope/log/logfile"


char *FirstFile ();     /* Picks first file out of ReadDir's list of files */
char *MyName ();        /* Reads THISSITEFILE to find out this site's name */
char *ReadStr ();       /* Reads a str from a file up to a NL              */
char *StrSave ();       /* Strcopy with malloc (see K&R p. 103)            */
char *StripMe ();       /* Gets first site from full path (a!b!c!d)        */

int  Annoy ();          /* Writes message(s) to operator (stderr)          */
int  CallHo ();         /* Call and login to the (ugh) Honeywell           */
int  CallS1 ();         /* Call and login to the (ugh) Series/1            */
int  DateTime ();       /* Gives date and time in a nice format            */
int  Dial ();           /* Initial modem-dialing module                    */
int  FRename ();        /* Changes the name of a file                      */
int  GetSeqNo ();       /* Gets the next sequence number from a msg queue  */
int  HandleLogin ();    /* Establishes connection with machine calling in  */
int  HangUp ();         /* Hangs up the modem                              */
int  OpenModem ();      /* Opens the modem line                            */
int  Read ();           /* Our own version of read(2) with timeouts, etc   */
int  ReadDir ();        /* Reads filenames in a directory                  */
int  ValidPath ();      /* Validates a site and gives the full path to it  */
int  ValidSite ();      /* Validates a site and gives the phone nums to it */
int  WriteLog ();       /* Writes message(s) to LOGFILE                    */
```

```
/* Unix calls.  Unix is a registered trademark of ATT Bell Laboratories   */

char *calloc ();
char *ctime ();
char *getenv ();
char *malloc ();
char *strcpy ();
int atoi ();
int chmod ();
int link ();
int unlink ();
long time ();
unsigned sleep ();
```

```
#include "net.h"


int HandleLogin (Fd)

int Fd;

/* Validates call-in connection from remote sender.      */
/* Returns 1 if validated.                               */

{
   char Ch, LoginLine [8];
   int  LoginStatus;
   register int Count, NumTries = 0;

   do {
       Count = Read (Fd, &Ch, 1);

       NumTries++;
fprintf (stderr,"timed out %d times tries=%d\r", Count,NumTries);

     } while ((Ch != '\r') && (NumTries <= MAXTRIES));

   if (NumTries > MAXTRIES)
      {
       fprintf (stderr, "HandleLogin: Timed out on initial read.\n");
       return (FALSE);
      }

   ReadStr (Fd, LoginLine);
fprintf (stderr, "HandleLogin: received '%s'\n", LoginLine);

   LoginStatus = (EQUALS (LoginLine, LOGINMSG));

   if (LoginStatus)
      write (Fd, ACKMSG, strlen (ACKMSG));
   else
      write (Fd, NACKMSG, strlen (NACKMSG));

   write (Fd, CR, 1); /* Xmit Carriage-Return */

   return (LoginStatus);
}
```

```
#include "net.h"


int ModemFd;
struct termio TTYSet;

int OpenModem ()

{
   int OFlag = O_RDWR | O_NDELAY;

   if ((ModemFd = open (MODEMLINE, OFlag)) == ERR)
      {
       fprintf (stderr, "OpenModem: Can't open %s\n", MODEMLINE);
       exit (1);
      }

    if (ioctl (ModemFd, TCGETA, &TTYSet) == ERR) /* get old modem settings */
       {
           fprintf (stderr, "OpenModem: Can't get old modem settings\n");
        exit (1);
       }


    TTYSet.c_iflag &= ~INPCK;          /* Don't check input parity    */
    TTYSet.c_iflag |= PARMRK;          /* Mark parity errors (??)     */
    TTYSet.c_iflag |= ISTRIP;          /* Get rid of bit 8            */

    TTYSet.c_cflag |= PARENB;
    TTYSet.c_cflag &= ~CSIZE;
    TTYSet.c_cflag |= CS7;
    TTYSet.c_cflag &= ~PARODD;         /* Set even parity             */
    TTYSet.c_cflag &= ~CBAUD;          /* Clear old baud-rate bits    */
    TTYSet.c_cflag |= B1200;           /* Set baud to 1200            */

    TTYSet.c_lflag &= ~ICANON;         /* Don't want canonical input  */
    TTYSet.c_lflag &= ~ECHO;           /* No echo                     */

    TTYSet.c_cc [VEOF] = '\001';       /* MIN = 1 char (no buffering) */

    if (ioctl (ModemFd, TCSETA, &TTYSet) == ERR) /* set new modem attributes */
       {
           fprintf (stderr, "OpenModem: Can't set new modem attributes\n");
        exit (1);
       }

    return (ModemFd);
}
```

```
#include "net.h"

/* returns 0 if timed out, 1 if got NumBytes chars, -1 if ERROR */

int Read (Fd, Ptr, NumBytes)

int  Fd;
char *Ptr;
unsigned  NumBytes;

{
   register int NumTries;
   register int ReadRes;


   for ( ; NumBytes > 0;.NumBytes--)
       {
        NumTries = 0;
        while (((ReadRes = read (Fd, Ptr, 1)) == 0) && (NumTries < MAXTRIES))
               NumTries++;

        if (NumTries >= MAXTRIES)
           return (0);

        if (ReadRes == ERR)
           return (ERR);

/* ignore nulls or parity-error indicators */

        if ((*Ptr != '\0') && (*Ptr != '\377'))
            Ptr++;
        else
{
if (*Ptr == '\377')
fprintf (stderr, "Read: parity error.\n");
            NumBytes++;
}
       }

   return (ReadRes);
}
```

```
#include "net.h"
#include <sys/dir.h>

int ReadDir (DirName, Files)

char *DirName;
char *Files [];

/*
   Places the sorted filenames of DirName in the array Files.
   The filenames in DirName starting with a '.' will be ignored.
   Returns the number of unignored files, or ERR if DirName can't be read.
*/

{
   struct direct DirEntry;
   register int Dir;
   register int Num = 0;

   if ((Dir = open (DirName, O_RDONLY)) == ERR)
       return (ERR);

   read (Dir, &DirEntry, sizeof (struct direct));     /* Skip . and .. */
   read (Dir, &DirEntry, sizeof (struct direct));

   while (read (Dir, &DirEntry, sizeof (struct direct)) != 0)
      {
       if ((DirEntry.d_ino != (ino_t) 0) && (DirEntry.d_name [0] != '.'))
          {
           Files [Num] = StrSave (DirEntry.d_name);
           Num++;
          }
      }

   close (Dir);

   if (Num > 0)
       Sort (Files, Num);

   return (Num);
}



Sort (v, n)

char *v[];
int n;

{
   register int Gap, i, j;
   char *Temp;
```

```
    for (Gap = n/2; Gap > 0; Gap /= 2)
        for (i = Gap; i < n; i++)
            for (j = i-Gap; j >= 0; j -= Gap)
                {
                 if (strcmp (v [j], v [j+Gap]) <= 0)
                     break;
                 Temp = v [j];
                 v [j] = v [j+Gap];
                 v [j+Gap] = Temp;
                }
}
```

```
#include "net.h"


char *ReadStr (Fd, String)

int  Fd;
char *String;

{
   char *Ptr;
   char *RetPtr;

   Ptr = String-1;
   do {
       Ptr++;

       if (Read (Fd, Ptr, 1) == 0)
#ifdef XDEBUG
{fprintf (stderr,"ReadStr: timed out.\n");
#endif
            return (NULL);  /* timed out */
#ifdef XDEBUG
}
#endif

      } while ((*Ptr != '\r') && (*Ptr != '\n'));

   *Ptr = '\0';

   RetPtr = malloc (strlen (String) + 1);
   strcpy (RetPtr, String);

   return (RetPtr);
}
```

```
#include "net.h"


int Receiver (Fd)

int Fd;

{
   char Path [128], FullPath [128], FirstSite [20], *NewPath;
   char MsgLine [128];
   char Dummy [128];
   char MsgFileName [40];
   char *RetVal;  /* was = MsgLine */
   int  Local = FALSE;
   FILE *MsgFileFd;
   register int LineCount = 0;


   do {
       ReadStr (Fd, Path);
      } while (strlen (Path) == 0);
#ifdef XDEBUG
fprintf (stderr,"Receiver: path is '%s'\n", Path);
#endif

   ReadStr (Fd, FullPath);
#ifdef XDEBUG
fprintf (stderr,"Receiver: fullpath is '%s'\n", FullPath);
#endif

   if ((NewPath = StripMe (Path, FirstSite)) == NULL)
      {
       Local = TRUE;
       sprintf (MsgFileName, "%s/M%.5d", LCLQUEUE, GetSeqNo (LCLQUEUE));
      }
   else
      {
       sprintf (MsgFileName, "%s/M%.5d", OUTQUEUE, GetSeqNo (OUTQUEUE));
      }

   if ((MsgFileFd = fopen (MsgFileName, "w")) == NULL)
      {
       fprintf (stderr, "Receiver: Can't create%s\n", MsgFileName);
       WriteLog ("Receiver: Can't create", MsgFileName, "", "");
       return (1);  /* FIX RETURN CODE */
      }

   WriteLog ("Receiver: receiving into", MsgFileName, "", "");

   if (Local)
      fprintf (MsgFileFd, "\n");
   else
      fprintf (MsgFileFd, "%s\n", NewPath);

   fprintf (MsgFileFd, "%s\n", FullPath);

   RetVal = ReadStr (Fd, MsgLine);
   while ((!EQUALS (MsgLine, EOMSG)) && (RetVal != NULL))
      {
#ifdef XDEBUG
```

```
fprintf (stderr, "Receiver: got '%s'\n", MsgLine);
#endif
        LineCount++;
        fprintf (MsgFileFd, "%s\n", MsgLine);
        ReadStr (Fd, MsgLine);
       }

    fclose (MsgFileFd);

    if (chmod (MsgFileName, 0666) == ERR)
       {
        fprintf (stderr, "Receiver: Can't chmod !!!\n");
        exit (1);
       }

    if (RetVal == NULL)
       {
        fprintf (stderr, "Receiver: timed out reading message body.\n");
        WriteLog ("Receiver: Timed out;", MsgFileName, "not received", "");
        unlink (MsgFileName);
        return;
       }

    sprintf (LineCountStr, "%d\r", LineCount);
    write (Fd, LineCountStr, strlen (LineCountStr));
#ifdef XDEBUG
fprintf (stderr, "Receiver: sent line count (%d)\n", LineCount);
#endif

    do {
        ReadStr (Fd, MsgLine);
       } while ( (!EQUALS (MsgLine, ACKMSG)) && (!EQUALS (MsgLine, NACKMSG)) );

    if (EQUALS (MsgLine, NACKMSG))
       {
        fprintf (stderr, "Receiver: (debug) Received NACK.\n");
        WriteLog ("Receiver: got NACK;", MsgFileName, "aborted", "");
        unlink (MsgFileName);
        return;
       }
#ifdef XDEBUG
fprintf (stderr, "Receiver: received ACK. Saved file %s.\n", MsgFileName);
#endif
    WriteLog ("Receiver:", MsgFileName, "received OK", "");

    if (Local)
       {
        StripMe (FullPath, FirstSite);
        Annoy ("Received from", FirstSite, "into", MsgFileName);
       }

    return (0);
}
```

```
#include "net.h"


int Sender (ModemFd)

int ModemFd;

{
   char MsgFileName [40];
   char *TempFileName;     /* Gets allocated by FirstFile */
   char FromModem [20];
   int  MsgFileFd;
   char Path [128];
   char FirstSite [20];
   char FullPath [128];
   char *TelNums [20];
   char Ch;
   int  DialProg;
   register int NumLines = 0;

   if ((TempFileName = FirstFile (OUTQUEUE)) != NULL)
      {
       FlushModemInput (ModemFd);

       sprintf (MsgFileName, "%s/%s", OUTQUEUE, TempFileName);

       if ((MsgFileFd = open (MsgFileName, O_RDONLY)) == ERR)
          {
           fprintf (stderr, "Sender: Can't open %s\n", MsgFileName);
           WriteLog ("Sender: Can't open msg file", MsgFileName, "", "");
           exit (1);
          }

       ReadStr (MsgFileFd, Path);
       ReadStr (MsgFileFd, FullPath);

       StripMe (Path, FirstSite);

       if (FirstSite == NULL)
          {
           fprintf (stderr, "Sender: Empty path in file %s\n", MsgFileName);
           WriteLog ("Sender: Empty path in", MsgFileName, "", "");
           exit (1);
          }

       if (ValidSite (FirstSite, &DialProg, TelNums))
          {
           if (Dial (ModemFd, DialProg, TelNums) == ERR)
              {
               fprintf (stderr, "Sender: Can't get through to %s\n", FirstSite);
               WriteLog ("Sender: Can't connect to", FirstSite, "", "");
               return (1);  /* FIX THIS RETURN CODE */
              }
           else
              {
#ifdef XDEBUG
fprintf (stderr, "Dialed ok, now about to send file...");
#endif
               WriteLog ("Sender: Connected to", FirstSite, "", "");
               write (ModemFd, Path, strlen (Path));
```

```
                write (ModemFd, CR, 1);
                write (ModemFd, FullPath, strlen (FullPath));
                write (ModemFd, CR, 1);

                while (read (MsgFileFd, &Ch, 1) != 0)
                    {
                     if (Ch == '\n')
                         {
                          NumLines++;
                          write (ModemFd, CR, 1);
                         }
                     else
                         write (ModemFd, &Ch, 1);
                    }

                FlushModemInput (ModemFd);

                write (ModemFd, EOMSG, strlen (EOMSG));
                write (ModemFd, CR, 1);
#ifdef XDEBUG
fprintf (stderr, "...done sending\n");
#endif

/* Editor's note: must take care of timeout here: */

                ReadStr (ModemFd, FromModem);

                while (strlen (FromModem) == 0)
                       {
                        ReadStr (ModemFd, FromModem);
                       }

                if (atoi (FromModem) != NumLines)
                    {
#ifdef XDEBUG
fprintf (stderr,"Sender: got '%s' from modem\n", FromModem);
#endif
                     write (ModemFd, NACKMSG, strlen (NACKMSG));
                     fprintf (stderr,
                           "Sender: Sending NACK - sent %d lines\n", NumLines);
                     WriteLog ("Sender:", MsgFileName, "not sent", "(rcvd NACK)");
                    }
                else
                    {
#ifdef XDEBUG
fprintf (stderr,"Sender: got '%s' from modem.\n", FromModem);
#endif
                     write (ModemFd, ACKMSG, strlen (ACKMSG));
                     unlink (MsgFileName);
                     WriteLog ("Sender:", MsgFileName, "sent OK", "");
                     Annoy ("Sent", TempFileName, "to", FirstSite);
                    }
                write (ModemFd, CR, 1);
#ifdef XDEBUG
fprintf (stderr,"Sender: hanging up modem.\n");
                HangUp (ModemFd, DialProg);
#endif
               }
          }
        else
           {
```

```
            fprintf (stderr, "Sender: no path to %s defined.\n", FirstSite);
            WriteLog ("Sender: No path to", FirstSite, "defined", "");
            return (1);  /* FIX RETURN CODE */
           }
      }
}
```

```
#include "net.h"

char *StripMe (Path, FirstSite)

char *Path, *FirstSite;

/* Removes the first site name from Path and returns a pointer to the */
/* rest of the path. If there is only one site name left in Path, a   */
/* NULL is returned. Path is not altered, and FirstSite points to the */
/* site name that has been stripped off the front of the path.        */

{
   register int i = 0;

   while ((Path [i] != SEPCHAR) && (Path [i] != '\0'))
      {
       FirstSite [i] = Path [i];
       i++;
      }

   FirstSite [i] = '\0';

   return ((Path [i] == NULL) ? NULL : Path + i + 1);
}
```

```
#include "net.h"

char *StrSave (Str)       /* Save a string somewhere */

char *Str;

{
   char *Ptr;

   Ptr = malloc (strlen (Str) + 1);
   strcpy (Ptr, Str);

   return (Ptr);
}
```

```
#include "net.h"

int ValidPath (Site, Path)

char *Site;
char *Path;

{
   FILE *PathTableFd;
   char SomeSite [128];
   register int c;

   if ((PathTableFd = fopen (PATHTABLEFILE, "r")) == NULL)
      {
       fprintf (stderr, "ValidPath: Can't open %s\n", PATHTABLEFILE);
       exit (1);
      }

   getc (PathTableFd);                              /* skip first colon */

   do {
       fscanf (PathTableFd, "%s", SomeSite);        /* get a site */
       if (EQUALS (SomeSite, Site))
          {
           getc (PathTableFd);                      /* read NL   */
           fscanf (PathTableFd, "%s", Path);
           fclose (PathTableFd);
           return (TRUE);
          }
       else
          {
           c = getc (PathTableFd);
           while ((c != ':') && (c != EOF))
               c = getc (PathTableFd);
          }

      } while (c != EOF);

   fclose (PathTableFd);
   return (FALSE);
}
```

```
#include "net.h"


int ValidSite (Site, DialProg, TelNums)

char *Site;
int  *DialProg;
char *TelNums[];

/* Validates the existence of Site in SITETABLEFILE and returns as  */
/* arguments the DialProg code and a list of pointers to telephone  */
/* numbers through which the Site can be accessed.                  */
/* Returns: TRUE if valid; FALSE otherwise.                         */


{
   FILE     *SiteTableFd;
   char     SomeSite [128];
   char     *p;
   register int c, i;
   char     TempBuf [20];

   if ((SiteTableFd = fopen (SITETABLEFILE, "r")) == NULL)
      {
       fprintf (stderr, "ValidSite: Can't open %s\n", SITETABLEFILE);
       exit (1);
      }

   getc (SiteTableFd);                              /* skip first colon */

   do {
       fscanf (SiteTableFd, "%s", SomeSite);        /* get a site */

       if (EQUALS (SomeSite, Site))
          {
           SkipEOL (SiteTableFd);                   /* read NL   */
           fscanf (SiteTableFd, "%d", DialProg);
           i = 0;
           SkipEOL (SiteTableFd);

           c = getc (SiteTableFd);
           while ((c != ':') && (c != EOF))
              {
               ungetc (c, SiteTableFd);
               fscanf (SiteTableFd, "%s", TempBuf);
               p = malloc (strlen (TempBuf)+1);
               strcpy (p, TempBuf);
               TelNums [i++] = p;
               SkipEOL (SiteTableFd);
               c = getc (SiteTableFd);
              }
           TelNums [i] = NULL;
           fclose (SiteTableFd);
           return (TRUE);
          }
       else
          {
           c = getc (SiteTableFd);
           while ((c != ':') && (c != EOF))
               c = getc (SiteTableFd);
```

```
            }

        } while (c != EOF);

    fclose (SiteTableFd);
    return (FALSE);
}
```

```
#include "net.h"

int WriteLog (P1, P2, P3, P4)

char *P1, *P2, *P3, *P4;

{
   FILE *LogFd;
   char Date [26];

   DateTime (Date);

   if ((LogFd = fopen (LOGFILE, "a")) == NULL)
      {
       fprintf (stderr, "WriteLog: Can't open %s!  Good bye\n", LOGFILE);
       exit (1);
      }

   setbuf (LogFd, (char *) NULL);

   fprintf (LogFd, "%s %s %s %s %s\n", Date, P1, P2, P3, P4);

   fclose (LogFd);

   return (0);

}
```

```
#include "net.h"


main ()

{
   int ModemFd;
   char Ch;

   setbuf (stderr, (char *) NULL);


   ModemFd = OpenModem ();   /* must check for errors later */

   FlushModemInput (ModemFd);

   WriteLog ("Transceiver activated", "and ready", "", "");

   for ( ; ; )
       {
        if (read (ModemFd, &Ch, 1) == 0)
           {
fprintf (stderr, "starting sender...\n");
            Sender (ModemFd);
           }
        else
           {
#ifdef XDEBUG
fprintf (stderr, "starting handlelogin...\n");
#endif
            if (HandleLogin (ModemFd))
               {
#ifdef XDEBUG
fprintf (stderr, "starting receiver...\n");
#endif
                Receiver (ModemFd);
               }
           }
        sleep (4);
       }
}
```

Honeywell DPS/6 System Source Code

PASCAL programs

^ZSYS72>UDD>GOLDBERG>RECEIVER.PS

1985/01/07 1648:12.8

```
Program Receiver(input, output, messg, seq);


Type
     nametype  = packed array[1..5] of char;

(* Constant Declarations *)

Const
     ack       = 'ACK ';
     nack      = 'NACK';
     path1c1   = '>UDD>LOCMSG>MESNUM';
     pathnlo   = '>UDD>NLOMSG>MESNUM';
                 (* The files both named MESNUM in different directories
                    contains the sequence number of the last message for
                    local and no. local messages respectively. *)
     path1ms   = '>UDD>LOCMSG>M';
     pathnlm   = '>UDD>NLOMSG>M';


(* Var Declarations *)

Var
   msg         : text;
   messg       : text;
   seq         : text;
                 (* There is only one internal file that contains the
                    sequence number of the message. It would assigned
                    appropriately to one of the two external files. *)
   local       : boolean;  (* Indicates whether it's or not a local msg.
   i,j         : integer;
   ch          : char;
   ctln        : integer;  (* Counter of lines *)
   name        : nametype;
                 (* Contains the suffix of the name of the file in which
                    the message will be stored. *)
   answer      : packed array[1..4] of char;


(* End of global declarations. *)

(* *)

Procedure procheader(var local : boolean);

(* This procedure processes the header of the message and det.rmines
   whether it's a local or a non local message. *)

Const
    honeywell  = 'honeywell';
    hl         = 9;

var
    i,j        : integer;
```

```
    ch          : char;
    path        : packed array[1..80] of char;
    honeymaybe  : packed array[1..9] of char;

Begin
 reset(msg);
 if not eof(msg) then
    begin
     i := 0;
     read(msg,ch);
     while not eoln(msg) do  (* process the first line
                                of the header. *)
        begin
           i := i + 1;
           path[i] := ch;
           read(msg,ch)
        end;
     i := i + 1;
     path[i] := ch
   end;
  if i >= h1
    then
      begin
         for j := 1 to h1 do
             begin
                honeymaybe[j] := path[j]
             end
      end;
  if (i = h1) and (honeymaybe = honeywell)
     then local := true    (* it's a local mesage *)
     else
          local := false   (* it's not local *)
  end;  (* end of procedure process header. *)

 (* *)

 Procedure copy(var ctln : integer);
 (* Makes a copy of the message upon receiving it in an internal file.*)
 const
      zero        = 0;
      endofm      = '?MATZOH?';
 var
      i           : integer;
      ch          : char;
      nch         : integer;
      mat         : packed array[1..8] of char;
 (* End of declarations *)
begin
     rewrite(msg);
     endmess := false;
     ctln := zero;
     while not endmess do
     begin
          nch := zero;
          read(ch);
          while not eoln do
          begin
               nch := nch + 1;
               if nch <= 8
                  then mat[nch] := ch;
```

```
                write(msg,ch);
                read(ch)
            end; (* end of inner while loop. *)
            nch := nch + 1;
            if nch <= 8
               then
                  begin
                       mat[nch] := ch
                  end;
            if nch = 8
               then
                 begin
                      if mat = endofm
                         then (* the end of the message is reached *)
                             endmess := true
                 end;
            ctln := ctln + 1;
            readln;
            writeln(msg,ch)
          end;
     ctln := ctln - 3  (* Don't count the two lines of the
                          header and the line of ?MATZOH? *)
end; (* end of procedure copy *)


Procedure nameit(var name : nametype; local : boolean);
var
   ch          : char;
   messagnum   : integer;
   cnt         : integer;
   pname       : nametype;
   i,j         : integer;

begin
      if local
               then reset(seq,path1c1)
               else reset(seq,pathn1o);
         (* now seq is assigned to the appropriate external file. *)
      if not eof(seq)
         then
          begin
               read(seq,messagnum);        (* read an integer. *)
               messagnum := (messagnum + 1) mod 32767
          end;
      if local
               then rewrite(seq,path1c1)
               else rewrite(seq,pathn1o);
         (* now seq is assigned appropriately for writing .*)
      writeln(seq,messagnum);
      if local
               then reset(seq,path1c1)
               else reset(seq,pathn1o);
         (* once again, seq is assigned appropriately for reading *)
      cnt := 0;
      if not eof(seq)
        then
          begin
               read(seq,ch);
               while (not eoln(seq)) and (cnt < 5) do
               begin
```

```
                if ch <> ' '
                 then
                  begin
                       cnt := cnt + 1;
                       pname[cnt] := ch
                  end;
                read(seq,ch)
             end
      end;
   if cnt < 5
      then
       begin
            cnt := cnt + 1;
            pname[cnt] := ch
       end;
   i := 5;
   for j := cnt downto 1 do
   begin name[i] := pname[j]; i := i - 1 end;
   for j := i downto 1 do name[j] := '0'
end; (* end of procedure name it. *)


Procedure copyex(local : boolean; ctln : integer);
(* Copies the intenal message file into an external file in the appropri.
   directory local or non local messages. *)
var
   i        : integer;
   ch       : char;

begin
     reset(msg);
     if local
        then
         begin
              rewrite(messg, pathlms, name);
              readln(msg);
              writeln(messg)   (* When local, skip line from msg
                                  and write a blank lin. to messg. *)
         end
        else
         begin
              rewrite(messg, pathnlm, name);
              for i := 1 to 10 do read(msg,ch);
              read(msg,ch);
              while not eoln(msg) do
              begin
                   write(messg,ch);
                   read(msg,ch)
              end;
              writeln(messg,ch);
              readln(msg)
         end; (* The first line of header is copied into
                 the external file after having stripped
                 the word 'honeywell!' *)
        (*  Now messg is associated with the appropriate file. *)
     for i := 1 to ctln do
     begin
          read(msg,ch);
          while not eoln(msg) do
          begin
```

```
                write(messg,ch);
                read(msg,ch)
            end; (* end of while loop. *)
            writeln(messg,ch);
            readln(msg)
        end  (* end of outer for loop. *)
end; (* End of copyex procedure. *)



(*  MAIN *)
Begin
        writeln('ACK');      (* Acknowledge the sender of readiness. *)
        copy(ctln);  (* make a copy of the message in an internal file. *)
        procheader(local);  (* Process the message header. *)
        writeln(ctln);       (* Acknowledge the sender of the number of lines
                                received. *)
        ( * Hopefully will get an answer from sender saying OK. *)
        i := 0;
        read(ch);
        while (not eoln) and (i <= 4) do
        begin
             i := i + 1;
             answer[i] := ch;
             read(ch)
        end;
        i := i + 1;
        answer[i] := ch;
        if i < 4
           then for j := i + 1 to 4 do answer[j] := ' ';
        (* we completed with blanks the answer. *)
        if answer = ack
           then      (* Everything went fine so far, therefore we should copy
                        the mesage into an external file. *)
            begin
                     (* first we should create a name for it. *)
                nameit(name,local);
                copyex(local,ctln + 1)
            end
        (* else *)  (* it must be that the sender is not
                        happy with the number of lines I
                        received. Don't do anything. *)
end. (* This the end of the receiver program. *)
```

^ZSYS72>UDD>MIRI>SRC>DIAL.PS

1985/01/07 1644:44.8

```
(*$NOMAIN*)
procedure dial;external;
procedure dial;
const modem_address = '!DIAL03';
type  stat = set of 'A'..'Z';
      phone = packed array [1..7] of char;
var   ch    : char;
      command : packed array [1..7] of char;
      answer : packed array [1..8] of char;
      route : array [1..10] of phone;
      i,j : integer;
      ch1 : integer;
      status :  stat;
procedure sleep; nonpascal;
begin
   for i := 1 to 8 do
      answer [i] := ' ';
   close (output);
(* set modem status *)
   rewrite (output,modem_address);
   writeln ('   xy');
   command[1] := ' ';
   command[2] := chr(14);
   command[3] := 'N';
   command[4] := 'E';
   command[5] := 'W';
   command[6] := ' ';
   command[7] := '/';
   writeln (command);
   writeln (' /P 24');

   writeln (' /D ''T5179''');
   reset (input,modem_address);
   close (output);
   status := ['B','D','N','R','W','X'];
   REPEAT
      read (ch);
   UNTIL ((ch in status) or (ch = '/'));
   case ch of
   '/' : begin
            read (ch);                                  (*mod*)
            writeln (' Modem status is : ',ch);
         end;
   'X' : begin                                           (**)
            writeln (' Dial tone not detected.');
         end;                                            (**)

   end;
   close (output);

   sleep ;  (* sleep for 5 sec *)

(* rewrite (output,modem_address);
   close (output);                          *)
```

```
close (input);                (**)
end;
```

```
^ZSYS72>UDD>MIRI>SRC>SENDER.PS

1985/01/07 1645:39.6


program file_read;
const modem address = '!DIAL03';
      acklength = 3;
      escape_char = '/';
type  ltr = 'A'..'Z';
var   ch    : char;
      inbuffer : text;
(*    file_name : packed array [1..21] of char;       *)
      file_name : packed array [1..10] of char;
      i  : integer;
      letter : set of ltr;
      line_number : packed array [1..10] of char;
      line_count, line_count_rec : integer;
      phone : packed array [1..8] of char;
      ackmsg,accept : packed array [1..10] of char;
      success, flag : boolean;


   procedure dial;external;
   procedure hangup;external;

begin
   accept := 'ACK*******';
   for i := 1 to 10 do                          (* clear the name buffer *)
      file_name[i] := ' ';

   dial;

   rewrite (output);

   file_name := 'MBOX>DRAFT';

   REPEAT
(* close (input);
   reset (input, modem address);
   close (output);
   rewrite (output,modem_address);  *)

(* send 'login' string and check for acknowledgement *)

   close (output);                                  (**)

   success := false;
   while not (success) do
   BEGIN
      rewrite (output, modem_address);              (**)
      writeln ('      ');                           (**)
      writeln ('LOGIN');
      close (output);
(*    readln;            *)
      close (input);                                (**)
      reset (input, modem_address);                 (**)
      i := 1;
      while not eoln do
```

```
         begin
            read (ackmsg[i]);
            i := i + 1;
         end;
      if i <> (acklength+1) then success := false
      else
         begin
            success := true;
            i := 1;
            while (ackmsg[i] = accept[i]) do
               i:= i + 1;
         end;
      if (i = (acklength+1)) then
         success := true
      else
         success := false;
   END;

(* open the file with the name file.name and open the communication

   port/modem_port *)
   for i := 1 to 10 do
      line_number[i] := ' ';

   close (input);
   reset (input,file_name);
   rewrite (output,modem_address);

   line_count := 0;
   read (ch);
   while not eof do
      begin
         write (ch);
         if (ch = escape_char) then
            write (ch);
         if eoln then
            begin
               readln;
               line_count := line_count + 1;
               writeln;
            end;
         if (not eof) then
            read (ch);
      end;
  writeln ('?MATZOH?');

  line_count := line_count - 2;    (* Do not count first two lines *)

  close (input);
  reset (input,modem address);
  read (line_count_rec);
  if (line_count = line_count_rec) then
     writeln ('ACK')
  else
     writeln ('NACK');
  UNTIL (line_count = line_count_rec);
  close (output);
  hangup;
end.
```

Series/1 System Source Code

EDL programs

```
SENDER   PROGRAM  START
*****************************************************************
* PROGRAM TO SEND A MESSAGE FILE TO THE ONYX COMPUTER VIA MODEM.
* WRITTEN BY BILL PUTNAM  4/11/84
* THIS PROGRAM USES THE GENERIC SMARTMODEM SUBROUTINES IN MODSUBS,SRC
*****************************************************************
 PRINT OFF
 COPY TCBEQU
 COPY DSCBEQU
 COPY DDBEQU
 COPY PROGEQU
 COPY DSOPEN
 PRINT ON
MODEM    IOCB     $MODEM1
         DSCB     DS#=FILE,DSNAME=??
RECBUF   DC       256H'0'          DISK BUFFER FOR I/O
DISKBUFR EQU      RECBUF           DECLARED FOR DSOPEN
COUNT    DC       F'0'             NUMBER OF LINES SENT
#RCVD    DC       F'0'             NUMBER OF LINES RECEIVED
CONNMSG  DC       C'ACK '          CONNECTION MESSAGE
EOFMSG   TEXT     '?MATZOH?'       END OF TRANSMISSION MESSAGE
         EJECT
*****************************************************************
* SUBROUTINES FOR USING THE SMARTMODEM ON AN ACCA LINE.
* WRITTEN BY BILL PUTNAM  3/15/84
*  -THE ROUTINES EXPECT THE MODEM TO SEND NUMERIC RESULT CODES.
*  -ALL ROUTINES LEAVE THE MODEM DEQUEUED.
*  -ALL ROUTINES WILL ENQUEUE THE MODEM AS NEEDED, EXCEPT LISTEN
*   AND GETMODEM
*****************************************************************
MSG      TEXT     LENGTH=128   MESSAGE BUFFER USED BY SUBROUTINES
*****************************************************************
*   SUBROUT TO EBQUEUE THE MODEM
*        GADDR = ADDRESS TO GOTO IF TIMEOUT OCCURRS
* EXPECTS THE MODEM TO BE DEQUEUED, LEAVES IT ENQUEUED
*****************************************************************
         SUBROUT GETMODEM,GADDR
GTRY     DEQT
         ENQT MODEM,BUSY=GBUSY
GRET     RETURN
*
GBUSY    EQU *
         DEQT
         QUESTION '@MODEM IS BUSY.@TRY AGAIN ? ',YES=GTRY
         MOVE GETMODEM-2,GADDR        SET RETURN ADDRESS TO GADDR
         GOTO GRET
*****************************************************************
*   DIAL UP THE MODEM AT THE GIVEN PHONE NUMBER.
*       NODIAL IS THE ADDRESS TO GO TO IF THE MODEM CAN'T CONNECT
*****************************************************************
```

```
          SUBROUT DIAL,NODIAL
          PRINTEXT 'ATD5179'        DIAL COMMAND
          PRINTEXT SKIP=1           SEND CR/LF
          CALL LISTEN,20,NODIAL     READ THE RESULT CODE
DRET      RETURN
*****************************************************************
*    SEND MESSAGES TO REMOTE SYSTEM & PRINT RESPONSES.
*         STEXT = ADDRESS OF THE MESSAGE TO SEND
*         SECS  = TIMEOUT INTERVAL
*         SADDR = ADDRESS TO GOTO IF TIMEOUT OCCURRS
*****************************************************************
          SUBROUT SENDTXT,STEXT,SECS,SADDR
          MOVE S#1SAVE,#1
          MOVE #1,STEXT
          PRINTEXT (0,#1)             SEND TEXT
          PRINTEXT SKIP=1             SEND CR/LF
          CALL LISTEN,SECS,SADDR      LISTEN FOR RESPONSE
          MOVE #1,S#1SAVE
          RETURN
S#1SAVE   DC F'0'                     SAVE REGISTER #1 HERE
*****************************************************************
*    SEND A 72 BYTE LINE TO OTHER SYSTEM
*         SDTEXT = ADDRESS OF THE MESSAGE TO SEND
*         SDSECS = TIMEOUT INTERVAL
*         SDNUM  = BLOCK COUNTER
*         SDADDR = ADDRESS TO GOTO IF UNSUCESSFULL
*****************************************************************
          SUBROUT SENDDATA,SDTEXT,SDSECS,SDNUM,SDADDR
          MOVE S#1SAVE,#1
          MOVE #1,SDTEXT              COPY THE DATA TO XMIT BUFFER
          MOVE MSG,(0,#1),(128,BYTES)
          MOVE MSG-2,SHEADER          SET CHARACTER COUNT
          PRINTEXT MSG                SEND DATA
          PRINTEXT SKIP=1             SEND CR/LF
          MOVE #1,S#1SAVE             RESTORE #1
          RETURN
SHEADER   DC  X'8048'                 DUMMY TEXT HEADER
*****************************************************************
*    SUBROUT TO LISTEN FOR RESPONSE FROM REMOTE SYSTEM
*         LSECS = TIMEOUT INTERVAL
*         LADDR = ADDRESS TO GOTO IF TIMEOUT OCCURRS
* EXPECTS THE MODEM TO BE ENQUEUED, LEAVES IT DEQUEUED
*****************************************************************
          SUBROUT LISTEN,LSECS,LADDR
          STIMER LSECS,TIO,SECS           SET TIMEOUT INTERVAL
          READTEXT MSG,MODE=LINE          LISTEN FOR RESPONSE
          TCBGET RETCODE,$TCBCO           GET RETURN CODE
          STIMER RESET                    RESET INTERVAL TIMER
          IF (RETCODE,EQ,-5) THEN         TIMEOUT OCCURRED
             MOVE LISTEN-2,LADDR          SET RETURN ADDRESS TO LADDR
```

```
          ENDIF
          RETURN
*****************************************************************************
*   HANG UP THE PHONE
*         HADDR = ADDRESS TO GOTO IF THE MODEM CAN'T BE RESET
*****************************************************************************
          SUBROUT HANGUP,HADDR
HTRY      EQU *
          PRINTEXT '+++'
          TERMCTRL DISPLAY
          CALL LISTEN,10,HADDR   READ THE RESULT CODE
          IF (MSG,NE,C'0',BYTE),GOTO,HERR1
          PRINTEXT 'ATZ'
          PRINTEXT SKIP=1
          CALL LISTEN,10,HADDR   READ THE RESULT CODE
          IF (MSG,NE,C'0',BYTE),GOTO,HERR1
          DEQT  MODEM
          PRINTEXT 'MODEM AVAILABLE',SKIP=1
          TERMCTRL DISPLAY
          RETURN
*
HERR1     DEQT
*         PRINTEXT 'MODEM NOT IN COMMAND MODE.',SKIP=1
          EJECT
*****************************************************************************
START     EQU       *
*
*** SET THE FILE NAME AND OPEN THE FILE FOR READING
*
          MOVE FILE+$DSCBVOL,$PARM1+8,(6,BYTES)
          MOVE FILE+$DSCBNAM,$PARM1,(8,BYTES)
          CALL DSOPEN,(FILE)
          IF (FILE,NE,-1),GOTO,OPENERR
*
*** DIAL UP THE MODEM AT THE GIVEN PHONE NUMBER.
*
          CALL GETMODEM,(ALLOCERR)          ENQUEUE THE MODEM
          CALL DIAL,(DERR1)                 TELL THE MODEM TO DIAL
          PRINTEXT '     '                  SAY HELLO
          PRINTEXT SKIP=1
          PRINTEXT 'LOGIN'
          PRINTEXT SKIP=1
          PRINTEXT SKIP=1                   PUNCTUATE WITH A CR
          CALL LISTEN,10,(LERR1)            LISTEN FOR CONNECTION MESSAGE
          IF (MSG,NE,CONNMSG,3) THEN
             DEQT
             PRINTEXT 'BAD CONNECTION',SKIP=1
             GOTO STOP
          ENDIF
*
```

```
*** SEND FILE TO REMOTE SYSTEM
*
          POINT FILE,1
LOOP      READ FILE,RECBUF,1,0,END=EOF,ERROR=IOERR
SEND1     CALL SENDDATA,(RECBUF),10,RETCODE,(SERR1)
          IF (RETCODE,NE,COUNT),GOTO,TXERR
          ADD COUNT,1
SEND2     CALL SENDDATA,(RECBUF+128),10,RETCODE,(SERR2)
          IF (RETCODE,NE,COUNT),GOTO,TXERR
          ADD COUNT,1
          GOTO LOOP
EOF       CALL SENDTXT,(EOFMSG),10,(SERR3)
          CONVTD #RCVD,MSG
          SUBTRACT #RCVD,2
          IF (#RCVD,NE,COUNT) THEN
             PRINTEXT 'NACK'
             PRINTEXT SKIP=1
             MOVE RETCODE,100
          ELSE
             PRINTEXT 'ACK'
             PRINTEXT SKIP=1
          ENDIF
*
*** HANG UP THE PHONE
*
STOP      CALL HANGUP,(NOHANG)
*
*** END OF PROGRAM
*
          DEQT
          IF (RETCODE,NE,-1) THEN
             PRINTEXT '@@ERROR IN TRANSMISSION.'
             PRINTEXT '@PLEASE RETRANSMIT LATER.'
          ELSE
             PRINTEXT '@@TRANSMISSION COMPLETE.'
          ENDIF
ABORT     PROGSTOP -1,LOGMSG=YES,P1=RETCODE
          EJECT
*******************************************************************************
* ERROR TRAPS
*******************************************************************************
TXERR     EQU *
          IF (RETCODE,EQ,-90) THEN TARGET FILE IS FULL
             PRINTEXT '@THE TARGET FILE IS FULL.'
             PRINTEXT '@TRANSMISSION ABORTED.'
             GOTO EOF
          ELSE
             GOTO SERR1
          ENDIF
ALLOCERR  EQU *
```

```
          PRINTEXT '@ERROR ALLOCATING REMOTE DATA SET.'
          QUESTION '@TRY AGAIN ? ',YES=START,NO=EOF
OPENERR   EQU *
          PRINTEXT '@ERROR OPENING SOURCE FILE.'
          PRINTEXT '@ERROR NUMBER '
          PRINTNUM FILE
          QUESTION '@START OVER ? ',YES=START,NO=EOF
IOERR     DEQT
          PRINTEXT '@I/O ERROR READING SOURCE FILE.'
          GOTO STOP
NOHANG    EQU *
          PRINTEXT '@UNABLE TO HANG UP PHONE.'
          GOTO ABORT
DERR1     EQU *
          PRINTEXT '@UNABLE TO CONTACT REMOTE SYSTEM.'
          GOTO QUERY
LERR1     EQU *
          PRINTEXT '@UNABLE TO LOAD REQUESTED PROGRAM.'
          GOTO QUERY
SERR0     DEQT
          PRINTEXT '@ERROR SENDING FILE NAME'
          QUESTION '@TRY AGAIN ? ',YES=START,NO=QUERY
SERR1     DEQT
          PRINTEXT 'BLOCK NUMBER ',SKIP=2
          PRINTNUM COUNT
          PRINTEXT ' WAS NOT SENT CORRECTLY.'
          QUESTION '@TRY AGAIN ? ',YES=SEND1,NO=QUERY
SERR2     DEQT
          PRINTEXT 'BLOCK NUMBER ',SKIP=2
          PRINTNUM COUNT
          PRINTEXT ' WAS NOT SENT CORRECTLY.'
          QUESTION '@TRY AGAIN ? ',YES=SEND2,NO=QUERY
SERR3     DEQT
          PRINTEXT '@ERROR SENDING EOF MESSAGE'
          QUESTION '@TRY AGAIN ? ',YES=EOF,NO=QUERY
QUERY     EQU *
          QUESTION '@RESTART PROGRAM ? ',YES=START,NO=STOP
          ENDPROG
          END
```

```
GENMSG   PROGRAM  START,DS=((??))
MSG      TEXT     LENGTH=72
#LINES   DC       F'0'
FOUND    DC       F'0'
RECBUF   DC    128F'0'
*
SITELIST EQU *
         DC  C'HONEYWELL ',C'ONYX!HONEYWELL'
         DC  C'GROUCHO   ',C'ONYX!GROUCHO  '
LASTSITE EQU *
SITELEN  EQU 10
PATHLEN  EQU 14
TABLEN   EQU SITELEN+PATHLEN
*
DEBUG    ECB      -1
START    WAIT DEBUG
         MOVE MSG,C' ',(72,BYTES)
         READTEXT MSG,'TO: ',SKIP=1,MODE=LINE
         MOVEA #1,SITELIST
         DO WHILE,(#1,NE,LASTSITE),AND,(FOUND,EQ,0)
            IF ((0,#1),EQ,MSG,+SITELEN) THEN
               MOVE FOUND,-1
               MOVE RECBUF,(+SITELEN,#1),(+PATHLEN,BYTES)
               MOVE RECBUF+128,(+SITELEN,#1),(+PATHLEN,BYTES)
               WRITE DS1,RECBUF,1,0,EOF=EOF,ERROR=IOERR
               ADD #LINES,2
            ELSE
               ADD #1,+TABLEN
            ENDIF
         ENDDO
         IF (FOUND,NE,-1) THEN
            PRINTEXT '@@THAT SITE IS NOT DEFINED.'
            QUESTION 'TRY AGAIN ? ',YES=START,NO=STOP
         ENDIF
         PRINTEXT '@@PLEASE ENTER YOUR MESSAGE.'
         PRINTEXT '@TYPE A PERIOD AT THE START OF A NEW LINE TO END.'
         MOVE MSG,C' ',(72,BYTES)
         MOVE #1,0
         DO WHILE,(MSG,NE,C'.',BYTE)
            READTEXT MSG,': ',MODE=LINE,SKIP=1
            MOVE (RECBUF,#1),MSG,(72,BYTES)
            IF (#1,EQ,0) THEN
                MOVE #1,128
            ELSE
                MOVE #1,0
                WRITE DS1,RECBUF,1,0,EOF=EOF,ERROR=IOERR
                ADD #LINES,2
            ENDIF
            MOVE MSG,C' ',(72,BYTES)
         ENDDO
```

```
          IF (#1,EQ,128) THEN
             MOVE RECBUF+128,C' ',40
             WRITE DS1,RECBUF,1,0,ERROR=IOERR,EOF=EOF
             ADD #LINES,2
          ENDIF
*         LOAD #SENDER,DS=(DS1),EVENT=DEBUG
*         WAIT DEBUG
          PROGSTOP -1,LOGMSG=YES,P1=RETCODE
          EJECT
*
*** ERROR TRAPS
*
EOF       PRINTEXT '@@MESSAGE FILE IS FULL.'
          PRINTEXT '@ENLARGE IT AND TRY AGAIN.'
          GOTO STOP
*
IOERR     PRINTEXT '@@I/O ERROR WRITING MESSAGE FILE.'
          PRINTEXT '@PROGRAM ABORTED.'
          GOTO STOP
          ENDPROG
          END
```